

1 MEANS AND APPARATUS FOR A SCALEABLE
2 CONGESTION FREE SWITCHING SYSTEM WITH
3 INTELLIGENT CONTROL

4 **by**

5
6 **John Hesse and Coke Reed**
7

8 RELATED PATENT AND PATENT APPLICATIONS
9

10 The disclosed system and operating method are related to subject
11 matter disclosed in the following patents and patent applications that are
12 incorporated by reference herein in their entirety:

- 13 1. U.S. Patent application serial number 09/009,703 (approved but not
14 issued) entitled, "A Scaleable Low Latency Switch for Usage in an
15 Interconnect Structure", naming John Hesse as inventor;
16 2. U.S. Patent No. 5,996,020 entitled, A Multiple Level Minimum Logic
17 Network;
18 3. United States patent application serial no. 09/693,359 entitled,
19 "Multiple Path Wormhole Interconnect", naming John Hesse as
20 inventor;
21 4. United States patent application serial no. 09/693,357 entitled,
22 "Scalable Wormhole-Routing Concentrator", naming John Hesse and
23 Coke Reed as inventors;

- 1 5. United States patent application serial no. 09/693,603 entitled,
2 "Scaleable Interconnect Structure for Parallel Computing and Parallel
3 Memory Access", naming John Hesse and Coke Reed as inventors;
4 6. United States patent application serial no. 09/693,358 entitled,
5 "Scalable Interconnect Structure Utilizing Quality-Of-Service
6 Handling", naming Coke Reed and John Hesse as inventors; and
7 7. United States patent application serial no. 09/692,073 entitled,
8 "Scalable Method and Apparatus for Increasing Throughput in
9 Multiple Level Minimum Logic Networks Using a Plurality of
10 Control Lines" naming Coke Reed and John Hesse as inventors.

11 FIELD OF THE INVENTION

12 The present invention relates to a method and means of controlling an
13 interconnection structure applicable to voice and video communication
14 systems and to data/Internet connections. More particularly, the present
15 invention is directed to the first scalable interconnect switch technology with
16 intelligent control that can be applied to an electronic switch, and an optical
17 switch with electronic control.

18 BACKGROUND OF THE INVENTION

19 There can be no doubt that the transfer of information around the
20 globe will be the driving force for the world's economy in this century. The
21 amount of information currently transferred between individuals,
22 corporations and nations must and will increase substantially. The vital
23 question, therefore, is whether there will be an efficient and low cost
24 infrastructure in place to accommodate the massive amounts of information
25 that will be communicated between numerous parties in the near future. The

1 present invention, as set forth below, answers that question in the
2 affirmative.

3 In addition to the numerous communication applications, there are
4 numerous other applications enabling a wide variety of products including
5 massively parallel supercomputers, parallel workstations, tightly coupled
6 systems of workstations, and database engines. There are numerous video
7 applications including digital signal processing. The switching systems can
8 also be used in imaging including medical imaging. Other applications
9 include entertainment including video games and virtual reality.

10 The transfer of information, including voice data and video, between
11 numerous parties on a world-wide basis, depends on the switches which
12 interconnect the communication highways extending throughout the world.
13 Current technology, represented, for example, by equipment supplied by
14 Cisco, allows 16 I/O slots (accommodating, for example, the OC-192
15 protocol), which provides 160 GBS in total bandwidth. The number of I/O
16 slots can be increased by selective interconnection of existing Cisco
17 switches, but this results in substantially increased costs with a significant
18 decrease in bandwidth per port. Thus, although Cisco switches are currently
19 widely used, it is apparent that current technology, as represented by existing
20 Cisco products, will not be able to accommodate the increasing flood of
21 information that will be flowing over the world's communication highways.
22 A family of patent filings has been created by the assignee of the present
23 invention to alleviate the current and anticipated problems of
24 accommodating the massive amounts of information that will be transferred
25 between parties in the near future. To fully appreciate the substantial
26 advance of the present invention, it is necessary to briefly summarize the
27 prior incorporated inventions, all of which are incorporated herein by

reference and are the building blocks upon which the present invention stands.

One such system "A Multiple Level Minimum Logic Network" (MLML network) is described in U.S. Patent No. 5,996,020, granted to Coke S. Reed on November 30, 1999, ("Invention #1"), the teachings of which are incorporated herein by reference. Invention #1 describes a network and interconnect structure which utilizes a data flow technique that is based on timing and positioning of message packets communicating throughout the interconnect structure. Switching control is distributed throughout multiple nodes in the structure so that a supervisory controller providing a global control function and complex logic structures are avoided. The MLML interconnect structure operates as a "deflection" or "hot potato" system in which processing and storage overhead at each node is minimized. Elimination of a global controller and also elimination of buffering at the nodes greatly reduces the amount of control and logic structures in the interconnect structure, simplifying overall control components and network interconnect components while improving throughput and achieving low latency for packet communication.

More specifically, the Reed Patent describes a design in which processing and storage overhead at each node is greatly reduced by routing a message packet through an additional output port to a node at the same level in the interconnect structure rather than holding the packet until a desired output port is available. With this design the usage of buffers at each node is eliminated.

In accordance with one aspect of the Reed Patent, the MLML interconnect structure includes a plurality of nodes and a plurality of interconnect lines selectively connecting the nodes in a multiple level

structure in which the levels include a richly interconnected collection of rings, with the multiple level structure including a plurality of $J+1$ levels in a hierarchy of levels and a plurality of $C \cdot 2^K$ nodes at each level (C is an integer representing the number of angles where nodes are situated). Control information is sent to resolve data transmission conflicts in the interconnect structure where each node is a successor to a node on an adjacent outer level and an immediate predecessor to a node on the same level. Message data from an immediate predecessor has priority. Control information is sent from nodes on a level to nodes on the adjacent outer level to warn of impending conflicts.

The Reed Patent is a substantial advance over the prior art in which packets proceed through the interconnect structure based on the availability of an input port at a node, leading to the packet's terminal destination. Nodes in the Reed Patent could be capable of receiving a plurality of simultaneous packets at the input ports of each node. However, in one embodiment of the Reed Patent, there was guaranteed availability of only one unblocked node to where an incoming packet could be sent so that in practice, in this embodiment, the nodes in the Reed Patent could not accept simultaneous input packets. The Reed Patent, however, did teach that each node could take into account information from a level more than one level below the current level of the packet, thus, reducing throughput and achieving reduction of latency in the network.

A second approach to achieving an optimum network structure has been shown and described in U.S. Patent Application Serial No. 09/009,703 to John E. Hesse, filed on January 20, 1998. ("Invention #2" entitled: "A Scaleable Low Latency Switch for Usage in an Interconnect Structure"). This patent application is assigned to the same entity as is the instant

1 application, and its teachings are also incorporated herein by reference in
 2 their entirety. Invention #2 describes a scalable low-latency switch which
 3 extends the functionality of a multiple level minimum logic (MLML)
 4 interconnect structure, such as is taught in Invention #1, for use in computers
 5 of all types, networks and communication systems. The interconnect
 6 structure using the scalable low-latency switch described in Invention #2
 7 employs a method of achieving wormhole routing by a novel procedure for
 8 inserting packets into the network. The scalable low-latency switch is made
 9 up of a large number of extremely simple control cells (nodes) which are
 10 arranged into arrays at levels and columns. In Invention #2, packets are not
 11 simultaneously inserted into all the unblocked nodes on the top level (outer
 12 cylinder) of an array but are inserted a few clock periods later at each
 13 column (angle). By this means, wormhole transmission is desirably
 14 achieved. Furthermore, there is no buffering of packets at any node.
 15 Wormhole transmission, as used here, means that as the first part of a packet
 16 payload exits the switch chip, the tail end of the packet has not yet even
 17 entered the chip.

18 Invention #2 teaches how to implement a complete embodiment of the
 19 MLML interconnect on a single electronic integrated circuit. This single-
 20 chip embodiment constitutes a self-routing MLML switch fabric with
 21 wormhole transmission of data packets through it. The scalable low-latency
 22 switch of this invention is made up of a large number of extremely simple
 23 control cells (nodes). The control cells are arranged into arrays. The
 24 number of control cells in an array is a design parameter typically in the
 25 range of 64 to 1024 and is usually a power of 2, with the arrays being
 26 arranged into levels and columns (which correspond to cylinders and
 27 angles, respectively, discussed in Invention #1). Each node has two data

input ports and two data output ports wherein the nodes can be formed into more complex designs, such as “paired-node” designs which move packets through the interconnect with significantly lower latency. The number of columns typically ranges from 4 to 20, or more. When each array contains 2^J control cells, the number of levels is typically $J+1$. The scalable low-latency switch is designed according to multiple design parameters that determine the size, performance and type of the switch. Switches with hundreds of thousands of control cells are laid out on a single chip so that the useful size of the switch is limited by the number of pins, rather than by the size of the network. The invention also taught how to build larger systems using a number of chips as building blocks.

Some embodiments of the switch of this invention include a multicasting option in which one-to-all or one-to-many broadcasting of a packet is performed. Using the multicasting option, any input port can optionally send a packet to many or all output ports. The packet is replicated within the switch with one copy generated per output port. Multicast functionality is pertinent to ATM and LAN/WAN switches, as well as supercomputers. Multicasting is implemented in a straightforward manner using additional control lines which increase integrated circuit logic by approximately 20% to 30%.

The next problem addressed by the family of patents assigned to the assignee of the present invention expands and generalizes the ideas of inventions # 1 and #2. This generalization (Invention #3 entitled: “Multiple Path Wormhole Interconnect”) is carried out in United States Patent Application Serial No. 09/693.359. The generalizations include networks whose nodes are themselves interconnects of the type described in Invention #2. Also included are variations of Invention #2 that include a richer control

1 system connecting larger and more varying groups of nodes than were
 2 included in control interconnects in Inventions #1 and #2. The invention
 3 also describes a variety of ways of laying out FIFOs and efficient chip floor
 4 planning strategies.

5 The next advance made by the family of patents assigned to the same
 6 assignee as is the present invention is disclosed in United States patent
 7 Application, Serial No. 09/693,357, entitled "Scalable Worm Hole-Routing
 8 Concentrator," naming John Hesse and Coke Reed as inventors. ("Invention
 9 #4")

10 It is known that communication or computing networks are comprised
 11 of several or many devices that are physically connected through a
 12 communication medium, for example a metal or fiber optic cable. One type
 13 of device that can be included in a network is a concentrator. For example, a
 14 large-scale, time-division switching network may include a central switching
 15 network and a series of concentrators that are connected to input and output
 16 terminals of other devices in the switching network.

17 Concentrators are typically used to support multi-port connectivity to
 18 or from a plurality of networks or between members of plurality of
 19 networks. A concentrator is a device that is connected to a plurality of
 20 shared communication lines that concentrates information onto fewer lines.

21 A persistent problem that arises in massively parallel computing
 22 systems and in communications systems occurs when a large number of
 23 lightly loaded lines send data to a fewer number of more heavily loaded
 24 lines. This problem can cause blockage or add additional latency in present
 25 systems.

26 Invention #4 provides a concentrator structure that rapidly routes data
 27 and improves information flow by avoiding blockages, that is scalable

virtually without limit, and that supports low latency and high throughput. More particularly, this invention provides an interconnect structure which substantially improves operation of an information concentrator through usage of single-bit routing through control cells using a control signal. In one embodiment, message packets entering the structure are never discarded, so that any packet that enters the structure is guaranteed to exit. The interconnect structure includes a ribbon of interconnect lines connecting a plurality of nodes in non-intersecting paths. In one embodiment, a ribbon of interconnect lines winds through a plurality of levels from the source level to the destination level. The number of turns of a winding decreases from the source level to the destination level. The interconnect structure further includes a plurality of columns formed by interconnect lines coupling the nodes across the ribbon in cross-section through the windings of the levels. A method of communicating data over the interconnect structure also incorporates a high-speed minimum logic method for routing data packets down multiple hierarchical levels.

The next advance made by the family of patents assigned to the same assignee as is the present invention is disclosed in United States patent Application, Serial No. 09/693,603, entitled "Scalable Interconnect Structure for Parallel Computing and Parallel Memory Access," naming John Hesse and Coke Reed as inventors. ("Invention #5")

In accordance with Invention 5, data flows in an interconnect structure from an uppermost source level to a lowermost destination level. Much of the structure of the interconnect is similar to the interconnects of the other incorporated patents. But there are important differences; in invention #5, data processing can occur within the network itself so that data entering the

1 network is modified along the route and computation is accomplished within
2 the network itself.

3 In accordance with this invention, multiple processors are capable of
4 accessing the same data in parallel using several innovative techniques.

5 First, several remote processors can request to read from the same data
6 location and the requests can be fulfilled in overlapping time periods.

7 Second, several processors can access a data item located at the same
8 position, and can read, write, or perform multiple operations on the same
9 data item overlapping times. Third, one data packet can be multicast to
10 several locations and a plurality of packets can be multicast to a plurality of
11 sets of target locations.

12 A still further advance made by the assignee of the present invention
13 is set forth in U.S. Patent Application, Serial No. 09/693,358, entitled
14 "Scalable Interconnect Structure Utilizing Quality-of-Service Handling,"
15 naming Coke Reed and John Hesse as inventors ("Invention # 6").

16 A significant portion of data that is communicated through a network
17 or interconnect structure requires priority handling during transmission.

18 Heavy information or packet traffic in a network or interconnection
19 system can cause congestion, creating problems that result in the delay or
20 loss of information. Heavy traffic can cause the system to store information
21 and attempt to send the information multiple times, resulting in extended
22 communication sessions and increased transmission costs. Conventionally, a
23 network or interconnection system may handle all data with the same
24 priority, so that all communications are similarly afflicted by poor service
25 during periods of high congestion. Accordingly, "quality of service" (QOS),
26 has been recognized and defined, which may be applied to describe various
27 parameters that are subject to minimum requirements for transmission of

1 particular data types. QOS parameters may be utilized to allocate system
2 resources such as bandwidth. QOS parameters typically include
3 consideration of cell loss, packet loss, read throughput, read size, time delay
4 or latency, jitter, cumulative delay, and burst sizes. QOS parameters may be
5 associated with an urgent data type such as audio or video streaming
6 information in a multimedia application, where the data packets must be
7 forwarded immediately, or discarded after a brief time period.

8 Invention #6 is directed to a system and operating technique that
9 allows information with a high priority to communicate through a network
10 or interconnect structure with a high quality of service handling capability.
11 The network of invention #6 has a structure that is similar to the structures
12 of the other incorporated inventions but with additional control lines and
13 logic that give high QOS messages priority over low QOS messages.
14 Additionally, in one embodiment, additional data lines are provided for high
15 QOS messages. In some embodiments of Invention #6, an additional
16 condition is that the quality of service level of the packet is at least a
17 predetermined level with respect to a minimum level of quality of service to
18 descent to a lower level. The predetermined level depends upon the location
19 of the routing node. The technique allows higher quality of service packets
20 to outpace lower quality of service packets early in the progression through
21 the interconnect structure.

22 A still further advance made by the assignee of the present invention
23 is described in U.S. Patent Application, Serial No. 09/692,073, entitled
24 "Scalable Method and Apparatus for Increasing Throughput in Multiple
25 Level Minimum Logic Networks Using a Plurality of Control Lines,"
26 naming Coke Reed and John Hesse as inventors ("Invention #7").

1 In Invention #7, the MLML interconnect structure comprises a
2 plurality of nodes with a plurality of interconnect lines selectively coupling
3 the nodes in a hierarchical multiple level structure. The level of a node
4 within the structure is determined by the position of the node in the structure
5 in which data moves from a source level to a destination level, or
6 alternatively laterally along a level of the multiple level structure. Data
7 messages (packets) are transmitted through the multiple level structure from
8 a source node to one of a plurality of designated destination nodes. Each
9 node included within said plurality of nodes has a plurality of input ports and
10 a plurality of output ports, each node capable of receiving simultaneous data
11 messages at two or more of its input ports. Each node is capable of
12 receiving simultaneous data messages if the node is able to transmit each of
13 said received data messages through separate ones of its output ports to
14 separate nodes in said interconnect structure. Any node in the interconnect
15 structure can receive information regarding nodes more than one level below
16 the node receiving the data messages. In invention #7, there are more
17 control interconnection lines than in the other incorporated invention. This
18 control information is processed at the nodes and allows more messages to
19 flow into a given node than was possible in the other inventions.

20 The family of patents and patent applications set forth above, are all
21 incorporated herein by reference and are the foundation of the present
22 invention.

23 It is, therefore, an object of the present invention to utilize the
24 inventions set forth above, to create a scalable interconnect switch with
25 intelligent control that can be used with electronic switches, optical switches
26 with electronic control and fully optical intelligent switches.

1 It is a further object of the present invention to provide a first true
2 router control utilizing complete system information.

3 It is another object of the present invention to only discard the lowest
4 priority messages in an interconnect structure when output port overload
5 demands message discarding.

6 It is a still further object of the present invention to ensure that partial
7 message discarding is never allowed, and that switch fabric overload is
8 always prevented.

9 It is another object of the present invention to ensure that all types of
10 traffic can be switched, including Ethernet packets, Internet protocol
11 packets, ATM packets and Sonnet Frames.

12 It is a still further object of the present invention to provide an
13 intelligent optical router that will switch all formats of optical data.

14 It is a further object of the present invention to provide error free
15 methods of handling teleconferencing, as well as providing efficient and
16 economical methods of distributing video or video-on-demand movies.

17 It is a still further and general object of the present invention to
18 provide a low cost and efficient scalable interconnect switch that far exceeds
19 the bandwidth of existing switches and can be applied to electronic switches,
20 optical switches with electronic control and fully optical intelligent switches.

21 SUMMARY OF THE INVENTION

22 There are two significant requirements associated with implementing
23 a large Internet switch that are not feasible to implement using prior art.
24 First, the system must include a large, efficient, and scalable switch fabric,
25 and second, there must be a global, scalable method of managing traffic
26 moving into the fabric. The patents incorporated by reference describe

highly efficient, scalable MLML switch fabrics that are self routing and non-blocking. Moreover, in order to accommodate bursty traffic these switches allow multiple packets to be sent to the same system output port during a given time step. Because of these features, these standalone networks desirably provide a scaleable, self-managed switch fabric. In systems with efficient global traffic control that ensure that no link in the system is overloaded except for bursts, the standalone networks described in the patents incorporated by reference satisfy the goals of scalability and local manageability. But there are still problems that must be addressed.

In real-life conditions, global traffic management is less than optimal, so that for a prolonged time traffic can enter the switch in such a way that one or more output lines from the switch become overloaded. An overload condition can occur when a plurality of upstream sources simultaneously send packets that have the same downstream address and continue to do so for a significant time duration. The resulting overload is too severe to be handled by reasonable amounts of local buffering. It is not possible to design any kind of switch that can solve this overload condition without discarding some of the traffic. Therefore, in a system where upstream traffic conditions causes this overload to occur there must be some local method for equitably discarding a portion of the offending traffic while not harming other traffic. When a portion of the traffic is discarded it should be the traffic with low value or quality of service rating.

In the following description the term "packet" refers to a unit of data, such as an Internet Protocol (IP) packet, an Ethernet frame, a SONET frame, an ATM cell, a switch-fabric segment (portion of a larger frame or packet), or other data object that one desires to transmit through the system. The

1 switching system disclosed here controls and routes incoming packets of one
2 or more formats.

3 In the present invention, we show how the interconnect structures,
4 described in patents incorporated by reference, can be used to manage a
5 wide variety of switch topologies, including crossbar switches given in prior
6 art. Moreover, we show how we can use the technologies taught in the
7 patents incorporated by reference to manage a wide range of interconnect
8 structures, so that one can build a scaleable, efficient interconnect switching
9 systems that handle quality and type of service, multicasting, and trunking.
10 We also show how to manage conditions where the upstream traffic pattern
11 would cause congestion in the local switching system. The structures and
12 methods disclosed herein manage fairly and efficiently any kind of upstream
13 traffic conditions, and provide a scalable means to decide how to manage
14 each arriving packet while never allowing congestion in downstream ports
15 and connections.

16 Additionally, there are I/O functions that are performed by line card
17 processors, sometimes called network processors, and physical medium
18 attachment components. In the following discussion it is assumed that the
19 functions of packet detection, buffering, header and packet parsing, output
20 address lookup, priority assignment and other typical I/O functions are
21 performed by devices, components and methods given in common switching
22 and routing practice. Priority can be based on the current state of control in
23 switching system **100** and information in the arriving data packet, including
24 type of service, quality of service, and other items related to urgency and
25 value of a given packet. This discussion mainly pertains to what happens to
26 an arriving packet after it has been determined (1) where to send it, and (2)
27 what are its priority, urgency, class, and type of service.

1 The present invention is a parallel, control-information generation,
2 distribution, and processing system. This scalable, pipelined control and
3 switching system efficiently and fairly manages a plurality of incoming data
4 streams, and apply class and quality of service requirements. The present
5 invention uses scalable MLML switch fabrics of the types taught in the
6 incorporated inventions to control a data packet switch of a similar type or of
7 a dissimilar type. Alternately stated, a request-processing switch is used to
8 control a data-packet switch: the first switch transmits requests, while the
9 second switch transmits data packets.

10 An input processor generates a request-to-send packet when it
11 receives a data packet from upstream. This request packet contains priority
12 information about the data packet. There is a request processor for each
13 output port, which manages and approves all data flow to that output port.
14 The request processor receives all requests packets for the output port. It
15 determines if and/or when the data packet may be sent to the output port. It
16 examines the priority of each request and schedules higher priority or more
17 urgent packets for earlier transmission. During overload at the output port, it
18 rejects low priority or low value requests. A key feature of the invention is
19 the joint monitoring of messages arriving at more than one input port. It is
20 not important that there is a separate logic associated with each output port
21 or if the joint monitoring is done in hardware or software. What is important
22 is that there exists a means for information concerning the arrival of a packet
23 MA at input port A and information concerning the arrival of packet MB at
24 input port B to be jointly considered.

25 A third switch called the answer switch, is similar to the first, and
26 transmits answer packets from the request processors back to the requesting
27 input ports. During an impending overload at an output, a request can

1 harmlessly be discarded by the request processor. This is because the
 2 request can easily be generated again at a later time. The data packet is
 3 stored at the input port until it is granted permission to be sent to the output;
 4 low-priority packets that do not receive permission during overload can be
 5 discarded after a predetermined time. An output port can never become
 6 overloaded because the request processor will not allow this to happen.
 7 Higher priority data packets are permitted to be sent to the output port during
 8 overload conditions. During an impending overload at an output port, low
 9 priority packets cannot prevent higher priority packets from being sent
 10 downstream.

11 Input processors receive information only from the output locations
 12 that they are sending to; request processors receive requests only from input
 13 ports that wish to send to them. All these operations are performed in a
 14 pipelined, parallel manner. Importantly, the processing workload for a given
 15 input port processor and for a given request processor does not increase as
 16 the total number of I/O ports increases. The scalable MLML switch fabrics
 17 that transmit the requests, answers and data, advantageously maintain the
 18 same per-port throughput, regardless of number of ports. Accordingly, this
 19 information generation, processing, and distribution system is without any
 20 architectural limit in size.

21 The congestion-free switching system consists of a data switch **130**
 22 and a scalable control system that determines if and when packets are
 23 allowed to enter the data switch. The control system consists of the set of
 24 input controllers **150**, the request switch **104**, and the set of request
 25 processors **106**, the answer switch **108**, and the output controller **110**. In one
 26 embodiment, there is one input port controller, IC **150**, and one request
 27 processor, RP **106**, for each output port **128** of the system. Processing of

1 requests and responses (answers) in the control system occurs in overlapped
2 fashion with transmission of data packets through the data switch. While the
3 control system is processing requests for the most recently arriving data
4 packets, the data switch performs its switching function by transmitting data
5 packets that received positive responses during a previous cycle.

6 Congestion in the data switch is prevented by not allowing any traffic
7 into the data switch that would cause congestion. Generally stated, this
8 control is achieved by using a logical "analog" of the data switch to decide
9 what to do with arriving packets. This analog of the data switch is called the
10 request controller **120**, and contains a request switch fabric **104** usually with
11 at least the same number of ports as the data switch **130**. The request switch
12 processes small request packets rather than the larger data packets that are
13 handled by the data switch. After a data packet arrives at an input controller
14 **150**, the input controller generates and sends a request packet into the
15 request switch. The request packet includes a field that identifies the
16 sending input controller and a field with priority information. These
17 requests are received by request processors **106**, each of which is a
18 representative for an output port of the data switch. In one embodiment,
19 there is one request processor for each data output port.

20 One of the functions of the input controllers is to break up arriving
21 data packets into segments of fixed length. An input controller **150** inserts a
22 header containing the address **214** of the target output port in front of each of
23 the segments, and sends these segments into data switch **130**. The segments
24 are reassembled into a packet by the receiving output controller **110** and sent
25 out of the switch through an output port **128** of line card **102**. In a simple
26 embodiment that is suitable for a switch in which only one segment can be
27 sent through line **116** in a given packet sending cycle, the input controllers

1 make a request to send a single packet through the data switch. A request
2 processor either grants or denies permission to the input controller for the
3 sending of its packet into the data switch. In a first scheme, the request
4 processors grant permission to send only a single segment of a packet; in a
5 second scheme, the request processors grant permission for the sending of
6 all or many of the segments of a packet. In this second scheme the segments
7 are sent one after another until all or most of the segments have been sent.
8 The segments making up one packet might be sent continuously without
9 interruption, or each segment might be sent in a scheduled fashion as
10 described with **FIG. 3C**, thus allowing other traffic to be attended to. The
11 second scheme has the advantage that input controllers make fewer requests
12 and therefore, the request switch is less busy.

13 During a request cycle, a request processor **106** receives, zero, one, or
14 more request packets. Each request processor receiving at least one request
15 packet ranks them by priority and grants one or more requests and may deny
16 the remaining requests. The request processor immediately generates
17 responses (answers) and sends them back to the input controllers by means
18 of a second switch fabric (preferably an MLML switch fabric), called the
19 answer switch, **AS 108**. The request processors send acceptance responses
20 corresponding to the granted requests. In some embodiments, rejection
21 responses are also sent. In another embodiment, the requests and answers
22 contain scheduling information. The answer switch connects the request
23 processors to the input controllers. An input controller that receives an
24 acceptance response is then allowed to send the corresponding data packet
25 segment or segments into the data switch at the next data cycle or cycles, or
26 at the scheduled times. An input controller receiving no acceptances does
27 not send a data packet into the data switch. Such an input controller can

1 submit requests at later cycles until the packet is eventually accepted, or else
 2 the input controller can discard the data packet after repeated denied
 3 requests. The input controller may also raise the priority of a packet as it
 4 ages in its input buffer, advantageously allowing more urgent traffic to be
 5 transmitted.

6 In addition to informing input processors that certain requests are
 7 granted, the request processor may additionally inform request processors
 8 that certain requests are denied. Additional information may be sent in case
 9 a request is denied. This information about the likelihood that subsequent
 10 requests will be successful can include information on how many other input
 11 controllers want to send to the requested output port, what is the relative
 12 priority of other requests, and recent statistics regarding how busy the output
 13 port has been. In an illustrative example, assume a request processor
 14 receives five requests and is able to grant three of them. The amount of
 15 processing performed by this request processor is minimal: it has only to
 16 rank them by priority and, based on the ranking, send off three acceptance
 17 response packets and two rejection response packets. The input controllers
 18 receiving acceptances send their segments beginning at the next packet
 19 sending time. In one embodiment, an input controller receiving a rejection
 20 might wait a number of cycles before submitting another request for the
 21 rejected packet. In other embodiments, the request processor can schedule a
 22 time in the future for request processors to send segment packets through the
 23 data switch.

24 A potential overload situation occurs when a significant number of
 25 input ports receive packets that must be sent downstream through a single
 26 output port. In this case, the input controllers independently, and without
 27 knowledge of the imminent overload, send their request packets through the

request switch to the same request processor. Importantly, the request switch itself cannot become congested. This is because the request switch transmits only a fixed, maximum number of requests to a request processor and discards the remaining requests within the switch fabric. Alternately stated, the request switch is designed to allow only a fixed number of requests through any of its output ports. Packets above this number may temporarily circulate in the request switch fabric, but are discarded after a preset time, preventing congestion in it. Accordingly, associated with a given request, an input controller can receive an acceptance, a rejection, or no response. There are a number of possible responses including:

- send only one segment of the packet at the next segment sending time,
- send all of the segments sequentially beginning at the next sending time,
- send all of the segments sequentially beginning at a certain future time prescribed by the request processor,
- send the segments in the future with a prescribed time for each segment,
- do not send any segments into the data switch,
- do not send any segments into the data switch and wait at least for a specified amount of time before resubmitting the request, because either a rejection response is returned or no response is returned, indicating the request was lost on account of too many requests submitted to that request processor.

An input controller receiving a rejection for a data packet retains that data packet in its input buffer and can regenerate another request packet for

1 the rejected packet at a later cycle. Even if the input controller must discard
 2 request packets the system functions efficiently and fairly. In an illustrative
 3 example of extreme overloading, assume 20 input controllers wish to send a
 4 data packet to the same output port at the same time. These 20 input
 5 controllers each send a request packet to the request processor that services
 6 that output port. The request switch forwards, say, five of them to the
 7 request processor and discards the remaining 15. The 15 input controllers
 8 receive no notification at all, indicating to them that a severe overload
 9 condition exists for this output port. In a case where three of the five
 10 requests are granted and two are denied by the request processor, the 17
 11 input controllers that receive rejection responses or no responses can make
 12 the requests again in a later request cycle.

13 "Multiple choice" request processing allows an input controller
 14 receiving one or more denials to immediately make one or more additional
 15 requests for different packets. A single request cycle has two or more sub-
 16 cycles, or phases. Assume, as an example, that an input controller has five
 17 or more packets in its buffer. Assume moreover, that the system is such that
 18 in a given packet sending cycle, the input controller can send two packet
 19 segments through the data switch. The request processor selects the two
 20 packets with the highest-ranking priority and sends two requests to the
 21 corresponding request processors. Assume moreover, that the request
 22 processor accepts one packet and denies the other. The input controller
 23 immediately sends another request for another packet to a different request
 24 processor. The request processor receiving this request will accept or deny
 25 permission for the input controller to send a segment of the packet to the
 26 data switch. The input controller receiving rejections may thus be allowed
 27 to send second-choice data packets, advantageously draining its buffer,

1 whereas it otherwise would have had to wait until the next full request cycle.

2 This request-and-answer process is completed in the second phase of a
3 request cycle. Even though requests denied in the first round are held in the
4 buffer, other requests accepted in the first and second rounds can be sent to
5 the data switch. Depending on traffic conditions and design parameters, a
6 third phase can provide yet another try. In this way, input controllers are
7 able to keep data flowing out of their buffers. Therefore, in case an input
8 controller can send N packet segments through lines 116 of the data switch
9 at a given time, the input controller can make up to N simultaneous requests
10 to the request processors in a given request cycle. In case K of the requests
11 are granted, the input controllers may make a second request to send a
12 different set of $N-K$ packets through the data switch.

13 In an alternate embodiment, an input controller provides the request
14 processor with a schedule indicating when it will be available for sending a
15 packet into the data switch. The schedule is examined by the request
16 processor, in conjunction with schedule and priority information from other
17 requesting input processors and with its own schedule of availability of the
18 output port. The request processor informs an input processor when it must
19 send its data into the switch. This embodiment reduces the workload of the
20 control system, advantageously providing higher overall throughput.

21 Another advantage of the schedule method is that request processors are
22 provided with more information about all the input processors currently
23 wanting to send to the respective output port, and accordingly can make
24 more informed decisions as to which input ports can send at which times,
25 thus balancing priority, urgency, and current traffic conditions in a scalable
26 means.

1 Note that, on average, an input controller will have fewer packets in
2 its buffer than can be sent simultaneously into the data switch, and thus the
3 multiple-choice process will rarely occur. However and importantly, an
4 impending congestion is precisely the time when the global control system
5 disclosed herein is most needed to prevent congestion in the data switch and
6 to efficiently and fairly move traffic downstream, based on priority, type and
7 class of service, and other QOS parameters.

8 In embodiments previously described, if a packet is refused entry into
9 the data switch, then at a later time the input controller may resubmits the
10 request at a later time. In other embodiments, the request processor
11 remembers that the request has been sent and later grants permission to send
12 when an opportunity is available. In some embodiments, the request
13 processor only sends acceptance responses. In other embodiments, the
14 request processor answers all requests. In this case, for each request that
15 arrives at a request processor, the input controller gets an answer packet
16 from the request processor. In case the packet is denied, this information
17 could give a time segment T so that the request processor must wait for a
18 time duration T before resubmitting a request. Alternatively, the request
19 processor could give information describing the status of competing traffic
20 at the request processor. This information is delivered to all input
21 controllers, in parallel, by the control system and is always current and up to
22 date. Advantageously, an input controller is able to determine how likely a
23 denied packet will be accepted and how soon. Extraneous and irrelevant
24 information is neither provided nor generated. The desirable consequence of
25 this method of parallel information delivery is that each input controller has
26 information about the pending traffic of all other input controllers wishing to
27 send to a common request processor, and only those input controllers.

1 As an example, during an overload condition an input controller may
 2 have four packets in its buffer that have recently had requests denied. Each
 3 of the four request processors has sent information that will allow the input
 4 controller to estimate the likelihood that each of the four packets will be
 5 accepted at a later time. The input controller discards packets or
 6 reformulates its requests based on probability of acceptance and priority, to
 7 efficiently forward traffic through system **100**. The control system disclosed
 8 herein importantly provides each input controller with all the information it
 9 needs to fairly and equitably determine which traffic to send into the switch.
 10 The switch is never congested and performs with low latency. The control
 11 system disclosed here can easily provide scalable, global control for
 12 switches described in the patents incorporated by reference, as well as for
 13 switches such as the crossbar switch.

14 Input controllers make requests for data that is “at” the input
 15 controller. This data can be part of a message that has arrived while
 16 additional data from the message has yet to arrive, it can consist of whole
 17 messages stored in buffers at the input port or it can consist of segments of a
 18 message where a portion of the message has already been sent through the
 19 data switch. In the embodiments previously described, when an input
 20 controller makes a request to send data to the data switch, and the request is
 21 granted then the data is always sent to the data switch. So, for example, if
 22 the input controller has 4 data carrying lines into the data switch, it will
 23 never make requests to use 5 lines. In another embodiment, the input
 24 controller makes more requests than it can use. The request processors
 25 honor a maximum of one request per input controller. If the input controller
 26 receives multiple acceptances, it schedules one packet to be sent into the
 27 switch and on the next round makes all of the additional requests a second

1 time. In this embodiment, the output controllers have more information to
 2 base their decisions upon and are therefore able to make better decisions.
 3 However, in this embodiment, each round of the request procedure is more
 4 costly. Moreover, in a system with four lines from the input controllers to
 5 the data switch and where time scheduling is not employed, it is necessary to
 6 make at least four rounds of requests per data transmission.

7 Additionally, there needs to be a means for carrying out multicasting
 8 and trunking. Multicasting refers to the sending of a packet from one input
 9 port to a plural number of output ports. However, a few input ports
 10 receiving lots of multicast packets can overload any system. It is therefore
 11 necessary to detect excessive multicasting, limit it, and thereby prevent
 12 congestion. As an illustrative example, an upstream device in a defect
 13 condition can transmit a continuous series of multicast packets where each
 14 packet would be multiplied in the downstream switch, causing immense
 15 congestion. The multicast request processors discussed later detect overload
 16 multicasting and limit it when necessary. Trunking refers to the aggregation
 17 of multiple output ports connected to the same downstream path. A plurality
 18 of data switch output ports are typically connected downstream to a high-
 19 capacity transmission medium, such as an optical fiber. This set of ports is
 20 often referred to as a trunk. Different trunks can have different numbers of
 21 output ports. Any output port that is a member of the set can be used for a
 22 packet going to that trunk. A means of trunking support is disclosed herein.
 23 Each trunk has a single internal address in the data switch. A packet sent to
 24 that address will be sent by the data switch to an available output port
 25 connected to the trunk, desirably utilizing the capacity of the trunk medium.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a schematic block diagram showing an example of a generic system constructed from building blocks including input processors and buffers, output processors and buffers, network interconnect switches that are used for traffic management and control, and a network interconnect switch that is used for switching data to target output ports.

FIG. 1B is a schematic block diagram of input control units. **FIG. 1C** is a schematic block diagram of output control units. **FIG. 1D** is a schematic block diagram showing a system processor and its connections to the switching systems and external devices.

FIG. 1E is a schematic block diagram showing an example of a full system of the type shown in **FIG. 1A** where the request switch and data switch system are combined in a single component, which advantageously can simplify processing in certain applications, and reduce the amount of circuitry needed to implement the system.

FIG. 1F is a schematic block diagram showing an example of a full system of the type shown in **FIG. 1A** where the request switch, answer switch and data switch system are combined in a single component, which advantageously reduces the amount of circuitry needed to implement the system in certain applications.

FIGs. 2A through **2L** are diagrams showing formats of packets used in various components of the switching system and for various embodiments of the system.

FIGs. 3A and **3B** are diagrams showing formats of packets used in various components for time-slot reservation scheduling of packets. **FIG. 3C** is a diagram of a method of time slot reservation showing how input

1 processors request to transmit at specified time periods in the future, how the
2 request processor receives them, and how the request processor replies to the
3 requesting input processors informing them when they can send.

4 **FIG. 4A** is a schematic block diagram of input control units with
5 multicast capability. **FIG. 4B** is a schematic block diagram showing a
6 request controller with multicast capability. **FIG. 4C** is a schematic block
7 diagram showing a data switch with multicast capability.

8 **FIG. 5A** is a schematic block diagram showing an example of the
9 system in **FIG. 1** with an alternate means of multicast support in the control
10 system. **FIG. 5B** is a schematic block diagram showing an alternate means
11 of multicast support in the data switch fabric.

12 **FIG. 6A** is a generalized timing diagram showing overlapped
13 processing of major components of the control and switching system. **FIG.**
14 **6B** is a more detailed an example of timing diagram showing overlapped
15 processing of control system components.

16 **FIG. 6C** is a timing diagram that illustrates a multicast timing scheme
17 where multicast requests are made only at designated time periods.

18 **FIG. 6D** is a generalized timing diagram of an embodiment of a
19 control system that supports the time-slot reservation scheduling discussed
20 with **FIGs. 3A, 3B** and **3C**.

21 **FIG. 7** is a diagram showing configurable output connections of an
22 electronic switch to advantageously provide flexibility in dynamically
23 matching traffic requirements to physical embodiment.

24 **FIG. 8** is a circuit diagram of the bottom levels of an electronic
25 MLML switch fabric that supports trunking in the nodes.

1 **FIG. 9** is a schematic block diagram of a design that provides high
2 bandwidth by employing a plural number of data switches corresponding to
3 a single control switch.

4 **FIG. 10A** is a schematic block diagram showing multiple systems 100
5 connected in layers to a set of line cards to increase system capacity and
6 speed in a scalable manner.

7 **FIG. 10B** illustrates a modification of the system of **FIG. 10A** where
8 a plurality of output controllers is combined into a single unit.

9 **FIG. 11A** is a schematic block diagram of a twisted-cube data switch
10 with concentrators employed between the switches.

11 **FIG. 11B** is a schematic block diagram of a twisted-cube data switch
12 and a control system including a twisted cube.

13 **FIG. 11C** is a schematic block diagram of a twisted-cube system with
14 two levels of management.

15 **FIG. 12A** is a schematic diagram of a node that has two data paths
16 from the east and two data paths from north and two data paths to the west
17 and two data paths to the south.

18 **FIG. 12B** is a schematic block diagram that shows a plurality of data
19 paths from the east and to the west, with different paths for each of short,
20 medium, long and extremely long packets.

21 **FIG. 13A** is a timing diagram for nodes of the type illustrated in **FIG**
22 **12A**.

23 **FIG. 13B** is a timing diagram for nodes of the type illustrated in **FIG**
24 **12B**.

25 **FIG. 14** is a circuit diagram of a portion of a switch supporting the
26 simultaneous transmission of packets of different lengths, and connections

1 showing nodes in two columns and two levels of the MLML interconnect
2 fabric.

3 DETAILED DESCRIPTION

4 **FIG. 1** depicts a data switch **130** and control system **100** connected to
5 a plurality of line cards **102**. The line cards send data to the switch and
6 control system **100** through input lines **134** and receive data from the switch
7 and control system **100** through lines **132**. The line cards receive and send
8 data to the outside world through a plurality of externally connected input
9 lines **126** and output lines **128**. Interconnect system **100** receives and sends
10 data. All of the packets enter and leave the system **100** through the line
11 cards **102**. Data entering system **100** is in the form of packets of various
12 lengths. The J line cards are denoted by $LC_0, LC_1, \dots LC_{J-1}$.

13 The line cards perform a number of functions. In addition to
14 performing I/O functions pertaining to standard transmission protocols given
15 in prior art, the line cards use packet information to assign a physical output
16 port address **204** and quality of service (QOS) **206** to packets. The line cards
17 build packets in the format shown in **FIG. 2A**. The packet **200** consists of
18 the four fields: BIT **202**, OPA **204**, QOS **206**, and PAY **208**. The BIT field
19 is a one-bit field that is always set to 1 and indicates the presence of a
20 packet. The output address field, OPA **204**, contains the address of the
21 target output. In some embodiments, the number of target outputs is equal to
22 the number of line cards. In other embodiments; the data switch may have
23 more output addresses than the number of line cards. The QOS field
24 indicates the quality of service type. The PAY field contains the payload to
25 be sent through data switch **130** to the output controller **110** specified by the
26 OPA address. Generally stated, the incoming packet may be considerably

1 larger than the PAY field. Segmentation and reassembly (SAR) techniques
2 are used to subdivide the incoming packet into a plurality of segments. In
3 some embodiments, all of the segments are of the same length, in other
4 embodiments; segments may be of different lengths. Each segment is placed
5 in the PAY field of a series of transmissions of packets **200** through the data
6 switch. The output controller performs reassembly of the segments, and
7 forwards the complete packet downstream through the line card. By this
8 method, system **100** is able to accommodate payloads varying widely in
9 length. The line card generates the QOS field from information in the
10 header of the arriving packet. Information needed to construct QOS fields
11 may remain in the PAY field. If this is the case, system **100** can discard the
12 QOS field when it is no longer used, and a line card downstream can obtain
13 quality of service information from the PAY field.

14 **FIG. 2** shows the formatting of data in various packets.

15 **Table 1** gives a brief overview of the contents of the fields in the
16 packets.

17

ANS	Answer from the request processor to the input controller granting permission for the input controller to send the packet segments to the data switch DS 130.
BIT	A one-bit field that is set to 1 when there is data in the packet. When set to 0 the remaining fields are ignored.
IPA	Input port address.
IPD	Input port data, used by the input processor in deciding which packets to send to the request processors.
KA	Address of the packet KEY in the keys buffer 166. This address, along with the input port address, is a unique packet identifier.
NS	Number of segments of a given packet stored in the packet buffer. This number is decremented when a segment packet is sent from the packet buffer to the output port.
OPA	The output port address is the address of : The target output port, The output controller processor associated with the target output port, or The request processor associated with the target output port.
PAY	The field containing the payload.
PBA	Packet buffer address 162, where the packets are stored.
PS	A segment of the packet.
QOS	A quality-of-service value, or priority value, assigned to the packet by the line card.
RBA	Request buffer address, where a given request packet is stored.
RPD	Request processor data, used to determine which packets are allowed to be sent through the data switch.

Table 1

The line cards 102 send packet 200, illustrated in FIG. 2A, to an input controller 150 through transmission line 134. The input controllers are

1 denoted by IC0, IC1, ... ICJ-1. In this embodiment the number of input
2 controllers is set equal to the number of line cards. In some embodiments an
3 input controller may handle a plurality of line cards.

4 A listing of the functions performed by the input controllers and
5 output controllers provides an overview of the workings of the entire system.
6 The input controllers 150 perform at least the following six functions:

- 7 1. they break the long packets into segment lengths that can be
8 conveniently handled by the data switch,
- 9 2. they generate control information that they use and also control
10 information to be used by the request processors,
- 11 3. they buffer incoming packets,
- 12 4. they make requests to the request processor for permission to send
13 packets through the data switch,
- 14 5. they receive and process answers from request processors, and
15 6. they send packets through the data switch.

16 The output controllers 110 perform the following three functions:

- 17 1. they receive and buffer packets or segments from the data switch,
- 18 2. they reassemble segments received from the data switch into full data
19 packets to send to the line cards, and
- 20 3. they send the reassembled packets to the line cards.

21 The control system is made up of input controllers 150, request
22 controller 120, and output controller 110. Request controller 120 is made up
23 of request switch 104, a plurality of request processors 106, and answer
24 switch 108. The control system determines if and when a packet or segment
25 is to be sent into the data switch. Data switch fabric 130 routes segments
26 from input controllers 150 to output controllers 110. A detailed description
27 of the control and switching structures, and control methods follows.

1 The input controller does not immediately send an incoming packet P
2 on line **116** through the data switch to the output port designated in the
3 header of P. This is because there is a maximum bandwidth on path **118**
4 from the data switch to the output port leading to the target of P, and a
5 plurality of inputs may have packets to send to the same output port at one
6 time. Moreover there is a maximum bandwidth on path **116** from an input
7 controller **150** to data switch **130**, a maximum buffer space at an output
8 controller **110**, and a maximum data rate from the output controller to the
9 line card. Packet P must not be sent into the data switch at a time that would
10 cause an overload in any of these components. The system is designed to
11 minimize the number of packets that must be discarded. However, in the
12 embodiment discussed here, if it is ever necessary to discard a packet, the
13 discarding is done at the input end by the input controller rather than at the
14 output end. Moreover, the data is discarded in a systematic way, paying
15 careful attention to quality of service (QOS) and other priority values. When
16 one segment of a packet is discarded, the entire packet is discarded.
17 Therefore, each input controller that has packets to send needs to request
18 permission to send, and the request processors grant this permission.

19 When a packet P **200** enters an input controller through line **134**, the
20 input controller **150** performs a number of operations. Refer to **FIG. 1B** for
21 a block diagram of internal components of an exemplary input controller and
22 an output controller. Data in the form of a packet **200** illustrated in **FIG. 2A**
23 enters an input controller processor **160** from the line card. The PAY field
24 **208** contains the IP packet, Ethernet frame, or other data object received by
25 the system. The input controller responds to arriving packet P by generating
26 internally used packets and stores them in its buffers **162**, **164** and **166**.
27 There are numerous ways to store the data associated with incoming packet

1 P. A method presented in the present embodiment is to store the data
2 associated with P in three storage areas:

- 3 1. the packet buffer **162** that is used for storing input segments
4 **232** and associated information,
- 5 2. the request buffer **164**, and
- 6 3. the keys buffer **166**, containing KEYs **210**.

7 In preparing and storing data in the KEYs buffer 166, the input
8 controller processes routing and control information associated with arriving
9 packet P. This is the KEY 210 information that the input controller uses in
10 deciding which requests to send to the request controller 120. Data in the
11 form given in FIG. 2B are referred to as a KEYs 210 and are stored in the
12 keys buffer 166 at the KEY address. BIT field 202 is a one-bit-long field
13 that is set to 1 to indicate the presence of a packet. IPD field 214 contains
14 the control information data that is used by input controller 160 in deciding
15 what requests to make to request controller 120. The IPD field may contain
16 a QOS field 206 as a sub-field. Additionally, the IPD field may contain data
17 indicating how long the given packet has been in the buffer and how full the
18 input buffers are. The IPD may contain the output port address and other
19 information that the input controller processor uses in deciding what
20 requests to submit. The PBA field 216 is the packet buffer address field and
21 contains the physical location of the beginning of the data 220 associated
22 with packet P in message buffer 162. The RBA field 218 is the request
23 buffer address field that gives the address of the data associated with packet
24 P in the request buffer 164. The data stored at the address "key address" in
25 buffer 166 is referred to as the KEY because it is this data that is used by the
26 input controller processor in making all of its decisions concerning which
27 requests to submit to the request controller 120. In fact, the decision

1 regarding which request is to be sent to the request controller is based on the
2 contents of the IPD field. It is advisable that the KEYs are kept in a high-
3 speed cache of the input control unit 150.

4 Arriving Internet Protocol (IP) packets and Ethernet frames range
5 widely in length. A segmentation and reassembly (SAR) process is used to
6 break the larger packets and frames into smaller segments for more efficient
7 processing. In preparing and storing the data associated with a packet P in
8 the packet buffer 162, the input controller processor 160 first breaks up PAY
9 field 208 in packet 200 into segments of a predetermined maximum length.
10 In some embodiments, such as those illustrated in **FIG. 12A**, there is one
11 segment length used in the system. In other embodiments, such as those
12 with nodes as illustrated in **FIG. 12B**, there is a plurality of segment lengths.
13 The multiple segment length system requires a slightly different data
14 structure than the one illustrated in **FIG. 2**. One with ordinary skills in the
15 art will be able to make the obvious changes to the data structure to
16 accommodate multiple lengths. Packet data formatted according to **FIG. 2C**
17 is stored at location PBA 216 in the packet buffer 162. The OPA field 204
18 contains the address of the target output port of the data switch of the packet
19 P. The NS field 226 indicates the number of segments 232 needed to
20 contain the payload PAY 208 of P.

21 The KA field 228 indicates the address of the KEY of packet P; the
22 IPA field indicates the input port address. The KA field together with the
23 IPA field forms a unique identifier for packet P. The PAY field is broken
24 into NS segments. In the illustration, the first bits of the PAY field are
25 stored on the top of the stack and the bits immediately following the first
26 segment are stored directly below the first bits; this process continues until
27 the last bits to arrive are stored on the bottom of the stack. Since the payload

1 may not be an integral multiple of the segment length, the bottom entry on
2 the stack may be shorter than the segment length.

3 Requests packets **240** have the format illustrated in **FIG. 2D**.

4 Associated with packet P, input controller processor **160** stores request
5 packets in request buffer **164** at request buffer address RBA. Note that RBA
6 **218** is also a field in KEY **210**. The BIT field consists of a single bit that is
7 always set to 1 in the presence of data at that buffer location. The output
8 port address that is the target for packet P is stored in the output port address
9 field OPA **204**. The request processor data field RPD **246** is information
10 that is to be used by the request processor **106** in the decision of whether or
11 not to allow packet P to be sent to the data switch. The RPD field may
12 contain the QOS field **206** as a sub-field. It may contain other information
13 such as:

- 14 • how full the buffers are at the input port where the packet P is
15 stored,
- 16 • information concerning how long the packet P has been stored,
- 17 • how many segments are in the packet P,
- 18 • multicast information,
- 19 • schedule information pertaining to when the input controller
20 can send segments, and
- 21 • additional information that is helpful for the request processor
22 in making a decision as to whether or not grant permission to
23 the packet P to be sent to the data switch **130**.

24 The fields IPA **230** and KA **228** uniquely identify a packet, and are
25 returned by the request processor in the format of answer packet **250**, as
26 illustrated in **FIG. 2E**.

In FIG. 1A, there are multiple data lines 122 from each input controller IC 150 to request controller 120, and also multiple data lines 116 from each input controller to data switch 130. Notice also that there are multiple data lines 124 from request controller 120 to each input controller, and multiple data lines 118 from the data switch to each output controller 110. In an embodiment where no more than one input port 116 of the data switch has a packet for a given output port 118, data switch DS 130 may be a simple crossbar, and control system 100 of FIG. 1A is capable of controlling it in a scalable manner.

REQUEST TO SEND AT NEXT PACKET SENDING TIME

At request times $T_0, T_1, \dots, T_{\max}$, input controller 150 may make requests to send data into switch 130 at a future packet-sending time, T_{msg} . The requests sent at time T_{n+1} are based on recently arriving packets for which no request has yet been made, and on the acceptances and rejections received from the request controller in response to requests sent at times T_0, T_1, \dots, T_n . Each input controller IC_n desiring permission to send packets to the data switch submits a maximum of R_{\max} requests in a time interval beginning at time T_0 . Based on responses to these requests, IC_n submits a maximum of R_{\max} additional requests in a time interval beginning at time T_1 . This process is repeated by the input controller until all possible requests have been made or request cycle T_{\max} is completed. At time T_{msg} the input controllers begin sending to the data switch those packets accepted by the request processors. When these packets are sent to the data switch, a new request cycle begins at times $T_0 + T_{\text{msg}}, T_1 + T_{\text{msg}}, \dots, T_{\max} + T_{\text{msg}}$.

In this description, n^{th} packet sending cycle begins at the same time as the first round of the $(n+1)^{\text{st}}$ request cycle. In other embodiments, the n^{th}

1 packet sending cycle may begins before or after first round of the (n+1)st
2 request cycle.

3 At time T_0 , there are a number of input controllers 150 that have one
4 or more packets P in their buffers that are awaiting clearance to be sent
5 through the data switch 130 to an output controller processor 170. Each
6 such input controller processor 160 chooses the packets that it considers
7 most desirable to request to send through the data switch. This decision is
8 based on the IPD values 214 in the KEYs. The number of request packets
9 sent at time T_0 by an input controller processor is limited to a maximum
10 value, R_{\max} . These requests can be made simultaneously or serially, or
11 groups of requests can be sent in a serial fashion. More than J requests can
12 be made into switch of a type taught in Inventions #1, #2 and #3, with J rows
13 on the top level by inserting the requests in different columns (or angles in
14 the nomenclature of Invention #1). Recall that one can simultaneously insert
15 into multiple columns only if multiple packets can fit on a given row. This
16 is feasible in this instance, because the request packets are relatively short.
17 Alternatively, the requests can be simultaneously inserted into a concentrator
18 of the type taught in Invention #4. Another choice is to insert the packets
19 sequentially into a single column (angle) with a second packet directly
20 following a first packet. This is also possible with MLML interconnect
21 networks of these types. In yet another embodiment, the switch RS, and
22 possibly the switches AS and DS, contain a larger number of input ports
23 than there are line cards. It is also desirable in some cases that the number
24 of output columns per row in the request switch is greater than the number
25 of output ports per row in the data switch. Moreover, in case these switches
26 are of a type taught in incorporated patents, the switches can easily contain
27 more rows on their uppermost level than there are line cards. Using one of

1 these techniques, packets are inserted into the request switch in the time
2 period from T_0 to $T_0 + d_1$ (where d_1 is a positive value). The request
3 processors consider all of the requests received from time T_0 to $T_0 + d_2$
4 (where d_2 is greater than d_1). Answers to these requests are then sent back to
5 the input controllers. Based on these answers, the input controllers can send
6 another round of requests at time T_1 (where T_1 is a time greater than $T_0 +$
7 d_2). The request processors can send an acceptance or a rejection as an
8 answer. It may be the case that some requests sent in the time period from
9 T_0 to $T_0 + d_1$ do not reach the request processor by time $T_0 + d_2$. The request
10 processor does not respond to these requests. This non-response provides
11 information to the input controller because the cause of the non-response is
12 congestion in the request switch. These requests may be submitted at
13 another request sending time T_n before time T_{msg} or at another time after
14 T_{msg} . Timing is discussed in more detail in reference to **FIGs. 6A** and **6B**.

15 The request processors examine all of the requests that they have
16 received. For all or a portion of the requests, the request processors grant
17 permission to the input controllers to send packets associated with the
18 requests to the output controllers. Lower priority requests may be denied
19 entry into the data switch. In addition to the information in the request
20 packet data field RPD, the request processors have information concerning
21 the status of the packet output buffers **172**. The request processors can be
22 advised of the status of the packet output buffers by receiving information
23 from those buffers. Alternately, the request processors can keep track of this
24 status by knowledge of what they have put into these buffers and how fast
25 the line cards are able to drain these buffers. In one embodiment, there is
26 one request processor associated with each output controller. In other
27 embodiments, one request processor may be associated with a plurality of

1 output ports. In alternate embodiments a plurality of request processors are
2 located on the same integrated circuit; in yet other embodiments the
3 complete request controller **120** may be located on one or a few integrated
4 circuits, desirably saving space, packaging costs and power. In another
5 embodiment, the entire control system and data switch may be located on a
6 single chip.

7 The decisions of the request processors can be based on a number of
8 factors, including the following:

- 9 • the status of the packet output buffers,
- 10 • a single-value priority field set by input controllers,
- 11 • the bandwidth from the data switch to the output controllers,
- 12 • the bandwidth out of the answer switch AS, and
- 13 • the information in the request processor data field RPD **246** of
14 the request packet.

15 The request processors have the information that they need to make
16 the proper decisions as to which data to send through the data switch.
17 Consequently, the request processors are able to regulate the flow of data
18 into the data switch and into the output controllers, into the line cards, and
19 finally into output lines **128** to downstream connections. Importantly, once
20 the traffic has left the input controller traffic flows through the data switch
21 fabric without congestion. If any data needs to be discarded, it is low
22 priority data and it is discarded at the input controller, advantageously never
23 entering the switch fabric, where it would cause congestion and could harm
24 the flow of other traffic.

25 Packets desirably exit system **100** in the same sequence they entered
26 it; no data ever gets out of sequence. When the data packet is sent to the

1 data switch, all of the data is allowed to leave that switch before new data is
2 sent. In this way, segments always arrive at the output controller in
3 sequence. This can be accomplished in a number of ways including:

- 4 1. the request processor is conservative enough in its operation so
5 that it is certain that all of the data passes through the data
6 switch in a fixed amount of time,
- 7 2. the request processor can wait for a signal that all of the data
8 has cleared the data switch before allowing additional data to
9 enter the data switch,
- 10 3. the segment contains a tag field indicating the segment number
11 that is used by the reassembly process,
- 12 4. the data switch is a crossbar switch that directly connects an
13 input controller to an output controller, or
- 14 5. a data switch of the stair-step MLML interconnect type
15 disclosed in Invention #3 can advantageously be used because it
16 uses fewer gates than a crossbar, and when properly controlled,
17 packets can never exit from it out of sequence.

18 In cases (1) and (2) above, using a switch of a given size with no more
19 than a fixed number N of inserted packets targeted for a given output port, it
20 is possible to predict an upper limit on the time T that packets can remain in
21 that switch. Therefore, the request processors can guarantee that no packets
22 are lost by granting no more than N requests per output port in time unit T.

23 In the embodiment shown in **FIG 1A**, there are multiple lines from
24 the data switch to the output controller. In one embodiment, the request
25 processor can assign a given line to a packet so that all of the segments of
26 that packet enter the output controller on the same line. In this case, the
27 answer from the request processor contains additional information that is

used to modify the OPA field in the packet segment header. Additionally, the request processor can grant permission for the input controller to send all of the segments of a given packet without interruption. This has the advantages of:

- reducing the workload for the input controller in that a single request is generated and sent for all segments of a data packet,
- allowing the input controller to schedule the plurality of segments in one operation and be done with it, and
- there are fewer requests for the request processor to handle, allowing more time for it to complete its analysis and generate answer packets,

The assignment of certain output controller input ports requires that additional address bits be used in the header of the data packets. One convenient way to handle the additional address bits is to provide the data switch with additional input ports and additional output ports. The additional output ports are used to put data into the correct bins in the packet output buffers and the additional input ports can be used to handle the additional input lines into the data switch. Alternatively, the additional address bits can be resolved after the packets leave the data switch.

It should be noted that in the case of an embodiment utilizing multiple paths connecting the input and output controllers to the rest of the system, all three switches, RS 104, AS 108, and DS 130, can deliver multiple packets to the same address. Switches with the capability to handle this condition must be used in all three locations. In addition to the obvious advantage of increased bandwidth, this embodiment allows the request processors to make more intelligent decisions since they base their decisions on a larger data set. In a second embodiment, request processors advantageously can send a

1 plurality of urgent packets from one input controller IC_n with relatively full
2 buffers to a single output controller OC_m , while refusing requests from other
3 input controllers with less urgent traffic.

4 Referring also to **FIGs. 1B, 1C and 6A**, in the operation of system
5 **100** events occur at given time intervals. At time T_0 , there are a number of
6 input controller processors **160** that have one or more packets P in their
7 buffers ready to be sent through the data switch **130** to an output control
8 processor **170**. Each input controller processor with a packet not yet
9 scheduled to be sent to the data switch chooses one or more packets for
10 which it requests permission to send through the data switch to its
11 destination output port. This decision to grant the request at a given time is
12 generally based on the IPD values **214** in the **KEYs**. At time T_0 , each input
13 controller processor **160** that contains one or more such data packets sends a
14 request packet to the request controller **120** asking permission to send the
15 data packet to the data switch. The request is accepted or denied based on
16 the IPD field of the request packet. The IPD field may consist of or may
17 contain a "priority value". In case this priority value is a single number, the
18 sole job of the request processors is to compare these numbers. This priority
19 value is a function of the QOS number of the packet. But whereas the QOS
20 number of the packet is fixed in time, the priority value may change in time
21 based on a number of factors including how long a message has been in a
22 buffer in an input port. Request packet **240** associated with the chosen data
23 packet is sent into request controller **120**. Each of these requests arrives at
24 the request switch **104** at the same time. The request switch routes packets
25 **240** using their OPA field **204** to the request processor **106** associated with
26 the target output port of the packet. The request processor, RP **106**, ranks

1 and generates answer packets **250** that are sent back to the respective input
2 controller through the answer switch **108**.

3 In the general case, several requests may be targeted for the same
4 request processor **106**. It is necessary that the request switch **104** can deliver
5 multiple packets to a single target request processor **106**. The MLML
6 networks disclosed in the patents incorporated by reference are able to
7 satisfy this requirement. Given this property along with the fact that the
8 MLML networks are self-routing and non-blocking, they are the clear choice
9 for a switch to be used in this application. As the request packets **240** travel
10 through the request switch, the OPA field is removed; the packet arrives at
11 the request processor without this field. The output field is not required at
12 this point because it is implied by the location of the packet. Each request
13 processor examines the data in the RPD field **246** of each request it receives
14 and chooses one or more packets that it allows to be sent to the data switch
15 **130** at prescribed times. A request packet **240** contains the input port
16 address **230** of the input controller that sent the request. The request
17 processors then generate an answer packet **250** for each request, which is
18 sent back to the input processors. By this means, an input controller receives
19 an answer for each granted request. The input controller always honors the
20 answer it received. Alternately stated, if the request is granted, the
21 corresponding data packet is sent into the data switch; if not, the data packet
22 is not sent. The answer packet **250** sent from a request processor to an input
23 controller uses the format given in **FIG. 2E**. If the request is not granted, the
24 request processor may send negative answers to input controllers. This
25 information may include the busy status of the desired output port and may
26 include information that the input controller can use to estimate the
27 likelihood that a subsequent request will be successful. This information

could include the number of other requests sent, their priority, and how busy the output port has been recently. The information could also include a suggested time to resubmit the request.

At time T_1 , suppose that an input processor IC_n that has a packet in its buffer that was neither accepted nor rejected in the T_0 round and suppose moreover that in addition to packets accepted in the T_0 round IC_n is capable of sending additional data packets at time T_{msg} . Then at time T_1 , IC_n will make requests to send additional packets through the data switch at time T_{msg} . Once again, from among all the requests received, the request processors 106 pick packets that are allowed to be sent.

During the request cycles, the input controller processors 160 use the IPD bits in the KEYs buffer to make their decisions, and the request processors 106 used the RPD bits to make their choice. More about how this is done is given later in this description.

After the request cycles at times $T_0, T_1, T_3, \dots T_{max}$, have been completed, each accepted packet is sent to the data switch. Referring to **FIG 2C**, when the input controller sends the first segment of the winning packet into the data switch, the top payload segment 232 (the segment with the smallest subscript) is removed from the stack of payload segments. The non-payload fields, 202, 204, 226, 228 and 230 are copied and placed in front of the removed payload segment 232 to form a packet 260 with a format given in **FIG. 2F**. The input controller processor keeps track of which payload segments have been sent and which segments remain. This can be done by decrementing the NS field 226. When the last segment is sent, all of the data associated with the packet can be removed from the three input controller buffers, 162, 164 and 166. Each input port of the data switch receives either one or no segment packets 260 because no input

1 controller processor sent a second request after the first request was granted.
2 Each output port of the data switch either receives no packets or one packet,
3 because no output controller processor granted more than could be handled
4 by the output ports. When segment packets exit the data switch **130**, they
5 are sent to output controllers **110** that reassemble them into a standard
6 format. The reassembled packets are sent to the line cards for downstream
7 transmission.

8 Since the control system assures that no input port or output port
9 receives multiple data segments, a crossbar switch would be acceptable for
10 use as a data switch. Therefore, this simple embodiment demonstrates an
11 efficient method of managing a large crossbar in an interconnect structure
12 that has bursty traffic and supports quality and type of service. An
13 advantage of a crossbar is that the latency through it is effectively zero after
14 its internal switches have been set. Importantly, an undesirable property of
15 the crossbar is that the number of internal nodes switches grows as N^2 ,
16 where N is the number of ports. Using prior art methods it is impossible to
17 generate the N^2 settings for a large crossbar operating at the high speeds of
18 Internet traffic. Assume that the inputs of a crossbar are represented by rows
19 and output ports by the connecting columns. The control system **120**
20 disclosed above easily generates control settings by a simple translation of
21 the OPA field **204** in the segment packet **260** to a column address, which is
22 supplied at the row where the packet enters the crossbar. One familiar with
23 the art can easily apply this 1-to-N conversion, termed a multiplexer, to the
24 crossbar inputs. When the data packets from the data switch reach the target
25 output controller **110**, the output controller processor **170** can begin to
26 reassemble the packet from the segments. This is possible because the NS
27 field **226** gives the number of the received segment and the KA field **228**

1 along with the IPA addresses **230** form a unique packet identifier. Notice
2 that, in case there are N line cards, it may be desirable to build a crossbar
3 that is larger than N X N. In this way there may be multiple inputs **116** and
4 multiple outputs **118**. The control system is designed to control this type of
5 larger than minimum size crossbar switch.

6 While a number of switch fabrics can be used for the data switch, in
7 the preferred embodiment an MLML interconnect network of the type
8 described in the incorporated patents is used for the data switch. This is
9 because:

- 10 • for N inputs into the data switch, the number of nodes in the
- 11 switch is of order $N \cdot \log(N)$,
- 12 • multiple inputs can send packets to the same output port and the
- 13 MLML switch fabric will internally buffer them,
- 14 • the network is self routing and non-blocking,
- 15 • the latency is low, and
- 16 • given that the number of packets sent to a given output is
- 17 managed by the control system, the maximum time through the
- 18 system is known.

19 In one embodiment the request processor **106** can advantageously
20 grant permission for the entire packet consisting of multiple segments to be
21 sent without asking for separate permission for each segment. This scheme
22 has the advantages that the workload of the request processor is reduced and
23 the reassembly of the packet is simpler because it receives all segments
24 without interruption. In fact, in this scheme, the input controller **150** can
25 begin sending segments before the entire packet has arrived from the line
26 card **102**. Similarly, the output controller **110** can begin sending the packet

1 to the line card before all of the segments have arrived at the output
 2 controller. Therefore, a portion of the packet is sent out of a switch output
 3 line before the entire packet has entered the switch input line. In another
 4 scheme, separate permission can be requested for each packet segment. An
 5 advantage of this scheme is that an urgent packet can cut through a non-
 6 urgent packet.

7 PACKET TIME-SLOT RESERVATION

8 Packet time slot reservation is a management technique that is a
 9 variant of the packet scheduling method taught in a previous section. At
 10 request times $T_0, T_1, \dots, T_{\max}$, an input controller 150 may make requests to
 11 send packets into the data switch beginning at any one of a list of future
 12 packet-sending times. The requests sent at time T_{n+1} are based on recently
 13 arriving packets for which no request has yet been made, and on the
 14 acceptances and rejections received from the request processor in response
 15 to requests sent at times T_0, T_1, \dots, T_n . Each input controller IC_n desiring
 16 permission to send packets to the data switch submits a maximum of R_{\max}
 17 requests in a time interval beginning at time T_0 . Based on responses to these
 18 requests, IC_n submits a maximum of R_{\max} additional requests in a time
 19 interval beginning at time T_1 . This process is repeated by the input
 20 controller until all possible requests have been made or request cycle T_{\max} is
 21 completed. When the request cycles $T_0, T_1, \dots, T_{\max}$ are all completed the
 22 process of making requests begins with request cycles at times $T_0 + T_{\max}, T_1$
 23 $+ T_{\max} \dots, T_{\max} + T_{\max}$.

24 When input controller IC_n requests to send a packet through the data
 25 switch, IC_n sends a list of times that are available for injecting packet P into
 26 the data switch so that all of the segments of the packet can be sent

1 sequentially to the data switch. In case packet P has k segments, IC_n lists
2 starting times T such that it is possible to inject the segments of the packet at
3 the sequence of times T, T+1, ... T+k-1. The request processor either
4 approves one of the requested times or rejects them all. As before, all
5 granted requests result in the sending of data. In case all of the times are
6 rejected in the T₀ to T₀+d₁ time interval, then IC_n may make a request at a
7 later time to send P at any one of a different set of times. When the
8 approved time for sending P arrives, then IC_n will begin sending the
9 segments of P through the data switch.

10 This method has the advantage over the method taught in the previous
11 section in that fewer requests are sent through the request switch. The
12 disadvantages are: 1) the request processor must be more complicated in
13 order to process the requests; and 2) there is a significant likelihood that this
14 “all or none” request cannot be approved.

15 SEGMENT TIME-SLOT RESERVATION

16 Segment time-slot reservation is a management technique that is a
17 variant of the method taught in the previous section. At request times T₀, T₁,
18 ... , T_{max}, input controller **150** may make requests to schedule the sending of
19 packets into the data switch. However, this method differs from packet
20 time-slot reservation method in that the message need not be sent with one
21 segment immediately following another. In one embodiment, an input
22 controller provides the request processor with information indicating a
23 plurality of times when it is able to send a packet into the data switch. Each
24 input controller maintains a Time-Slot Available buffer, TSA **168**, that
25 indicates when it is scheduled to send segments at future time slots.
26 Referring also to **FIG. 6A**, each TSA bit represents one time period **620** that

1 a segment can be sent into the data switch, where the first bit of TSA
2 represents the next time period after the current time. In another
3 embodiment, each input controller has one TSA buffer for each path 116 that
4 it has into the data switch.

5 The TSA buffer content is sent to the request processor along with
6 other information including priority. The request processor uses this time-
7 available information to determine when the input controller must send the
8 packet into the data switch. **FIGs. 3A and 3B** are diagrams of request and
9 answer packets that contain a TSA field. Request packet 310 includes the
10 same fields as request packet 240 and additionally contains a Request Time
11 Slot Available field, RTSA 312. Answer packet 320 includes the same
12 fields as answer packet 250 and additionally contains an answer time slot
13 field, ATSA 322. Each bit of ATSA 322 represents one time period 620 that
14 a packet can be sent into the data switch, where the first bit of ATSA
15 represents the next time period after the current time.

16 **FIG. 3C** is a diagram that shows an example of the time-slot
17 reservation processing. Only one segment is considered in the example. A
18 request processor contains TSA buffer 332 that is the availability schedule
19 for the request processor. RTSA buffers 330 are request times received from
20 input controllers. Contents of the buffers are shown at time **t0**, which is the
21 start of the request processing for the current time period, and time **t0'**,
22 which is the completion of request processing. At time **t0** RPr receives two
23 request packets 310 from two input controllers, ICi and ICj. Each RTSA
24 field contains a set of one-bit subfields 302 representing time periods **t1**
25 through **t11**. The value 1 indicates that the respective input controller can
26 send its packet at the respective time period; the value 0 indicates that it
27 cannot. RTSA request 302 indicates that ICi can send a segment at times **t1**,

1 **t3, t5, t6, t10** and **t11**. The content of the RTSA field from ICj is also
2 shown. Time-slot available buffer, TSA **332**, is maintained in the request
3 processor. The TSA sub-field for time **t1** is 0, indicating that the output port
4 is busy at that time. Note that the output port can accept a segment at times
5 **t2, t4, t6, t9** and **t11**.

6 The request processor examines these buffers in conjunction with
7 priority information in the requests, and determines when each request can
8 be satisfied. Subfields of interest in this discussion are shown circled in
9 **FIG. 3C**. Time **t2** is the earliest time permissible that a packet can be sent
10 in the data switch, as indicated by 1 in TSA **332**. Both requests have 0 in
11 subfield **t2**, therefore, neither of the input controllers can take advantage of
12 it. Similarly, neither input controller can use time **t4**. Time **t6 334** is the
13 earliest time that the output port is available and can also be used by an input
14 controller. Both input controllers can send at time **t6** and the request
15 processor selects ICi as the winner based on priority. It generates an Answer
16 Time Slot field **340** that has 1 in subfield **306** at time **t6** and 0 in all other
17 positions. This field is included in the answer packet that is sent back to ICi.
18 The request processor resets subfield **t6 334** to 0 in its TSA buffer, which
19 indicates that no other request can be sent at that time. The request
20 processor examines the request from ICj and determines that time **t9** is the
21 earliest that the request from ICj can be satisfied. It generates response
22 packet **442** that is sent to ICj, and resets bit **t9** to 0 in its TSA buffer.

23 When ICi receives an answer packet it examines ATSA field **340** to
24 determine when the data segment is to be sent into the data switch. This is
25 time **t6** in this example. If it receives all zeros, then the packet cannot be
26 sent during the time duration covered by the subfields. It also updates its
27 buffer by (1) resetting its **t6** subfield to 0, and (2) shifting all subfields to the

1 left by one position. The former step means that time **t6** is scheduled, and
2 the latter step updates the buffer for use during the next time period, **t1**.
3 Similarly, each request buffer shifts all subfields to the left by one bit in
4 order to be ready for the requests received at time **t1**.

5 Segmentation-and-reassembly (SAR) is advantageously employed in
6 the embodiments taught in the present section. When a long packet arrives it
7 is broken into a large number of segments, the number depending on the
8 length. Request packet **310** includes field NS **226** that indicates the number
9 of segments. The request processor uses this information in conjunction
10 with the TSA information to schedule when the individual segments are
11 sent. Importantly, a single request and answer is used for all segments.
12 Assume that the packet is broken into five segments. The request processor
13 examines the ATSA field along with its own TSA buffer and selects five
14 time periods when the segments are to be sent. In this case ATSA contains
15 five 1's. The five time periods need not be consecutive. This provides a
16 significant additional degree of freedom in the solution for time-slot
17 allocation for packets of different lengths and priorities. Assume on average
18 there are 10 segments per arriving IP or Ethernet packet. A request must
19 therefore be satisfied for every 10 segments sent through the data switch.
20 Accordingly, the request-and-answer cycle can be about 8 or 10 times longer
21 than the data switch cycle, advantageously providing a greater amount of
22 time for the request processor to complete its processing, and permitting a
23 stacked (parallel) data switch fabric to move data segments in bit-parallel
24 fashion.

25 When urgent traffic is to be accommodated, in one embodiment the
26 request processor reserves certain time periods in the near future for urgent
27 traffic. Assume that traffic consists of high proportion of non-urgent large

1 packets (that are broken into many segments), and a small portion of shorter,
2 but urgent, voice packets. A few large packets could ordinarily occupy an
3 output port for a significant amount of time. In this embodiment, requests
4 pertaining to large packets are not always scheduled for immediate or
5 consecutive transmission, even if there is an immediate slot available.
6 Advantageously, empty slots are always reserved at certain intervals in case
7 urgent traffic arrives. Accordingly, when an urgent packet arrives it is
8 assigned an early time slot that was held open, despite the concurrent
9 transmission of a plurality of long packets through the same output port.

10 An embodiment using time-slot availability information
11 advantageously reduces the workload of the control system, providing
12 higher overall throughput. Another advantage of this method is that request
13 processors are provided with more information, including time availability
14 information for each of the input processors currently wanting to send to the
15 respective output port. Accordingly, the request processors can make more
16 informed decisions as to which input ports can send at which times, thus
17 balancing priority, urgency, and current traffic conditions in a scalable
18 means of switching-system control.

19 OVER-REQUESTING EMBODIMENT

20 In embodiments previously discussed, the input controller submits
21 requests only when it was certain that if the request is accepted it could send
22 a packet. Furthermore, the input controller honors the acceptance by always
23 sending the packet or segment at the permitted time. Thus the request
24 processor knows exactly how much traffic will be sent to the output port. In
25 another embodiment, the input controllers are allowed to submit more
26 requests than they are capable of supplying data packets for. So that when

1 there are N lines **116** from the input controller to the data switch, the input
2 controller can make requests to send M packets through the system even in
3 the case where M is greater than N. In this embodiment, there can be
4 multiple request cycles per data-sending cycle. When an input controller
5 receives a plurality of acceptance notices from the request processors, it
6 chooses to select up to N acceptances that it will honor by sending the
7 corresponding packets or segments. In case there are one or more
8 acceptances than an input controller will honor, then that input controller
9 will inform the request processors which acceptances will be honored and
10 which will not. In the next request cycle, input controllers that received
11 rejections send a second round of requests for packets that were not accepted
12 in the first cycle. The request processors send back a number of acceptances
13 and each request processor can choose additional acceptances that it will act
14 upon. This process continues for a number of request cycles.

15 After these steps complete, the request processors have permitted no
16 more than the maximum number of packets that can be submitted to the data
17 switch. This embodiment has the advantage that the request processors have
18 more information upon which to make their decisions and therefore, and
19 provided that the request processors employ the proper algorithm, they can
20 give more informed responses. The disadvantage is that the method may
21 require more processing and that the multiple request cycles must be
22 performed in no more than one data-carrying cycle.

23 SYSTEM PROCESSOR

24 Referring to **FIG. 1D**, a system processor **140**, is configured to send
25 data to and receive data from the line cards **102**, the input controllers **150**,
26 output controllers **110**, and request processors **106**. The system processor

communicates with external devices **190** outside of the system, such as an administration and management system. A few I/O ports **142** and **144** of the data switch, and a few I/O ports **146** and **148** of the control system, are reserved for use by the system processor. The system processor can use the data received from input controllers **150** and from request processors **106** to inform a global management system of local conditions and to respond to the requests of the global management system. Input controllers and output controllers are connected by path **152** that is a means for them to communicate with other. Additionally, connection **152** allows the system processor to send a packet to a given input controller **150** by sending it through the data switch to the connected output controller. The latter forwards the packet to the connected input controller. Similarly, connection **152** allows an output controller to send a packet to the system processor by first sending it through the connected input controller. The system processor can send packets to the control system **120** by means of I/O connections **146**. The system processor receives packets from the control system by means of connections **148**. Accordingly, the system processor **140** has transmit and receive capabilities with respect to each request processor **106**, input controller **150**, and output controller **110**. Some uses of this communication capability include receiving status information and from the input and output controllers and the request processors, and transmitting setup and operational commands and parameters to them in a dynamic fashion.

COMBINED REQUEST SWITCH AND DATA SWITCH

In the embodiment illustrated in **FIG. 1E**, there is a single device RP/OC_N **154** that performs the functions of both a request processor RP_N **106** and an output controller OC_N **110**. Also, there is a single switch RS/DS

1 156 that performs the functions of both the request switch RS 104 and the
2 data switch DS 130. The line cards 102 accept the data packets and perform
3 functions already described in this document. The input controllers 150 may
4 parse and decompose the packet into a plurality of segments and also
5 perform other functions already described in this document. The input
6 controllers then request permission to inject the packet or segments into the
7 data switch.

8 In a first embodiment, the request packet is of the form illustrated in
9 FIG. 2D. These request packets are injected into RS/DS switch 156. In one
10 scheme, these request packets are injected into the RS/DS switch at the same
11 time as the data packets. In another scheme, these packets are injected at
12 special request-packet-injection times. Since the request packets are
13 generally shorter than data packets, the multiple-length-packet switch
14 embodiment of a previous section can be advantageously used for this
15 purpose.

16 In a second embodiment, the request packet is also a segment packet
17 as illustrated in FIG. 2F. The input controller sends the first segment, S_0 , of
18 a packet through the RS/DS switch. When S_0 arrives at the request
19 processor section of RP/OC_N, the request processor decides whether to allow
20 the sending of the rest of the segments of the packet, and if the rest of the
21 segments are allowed, the request processor schedules the sending of those
22 segments. These decisions are made in much the same fashion as they were
23 made by the request processors in FIG. 1A. The answers to these decisions
24 are sent to the input controllers through the answer switch AS. In one
25 scheme, the request processor sends an answer only when it receives the first
26 segment of a packet. In another scheme, the request processor sends an
27 answer to each request. In one embodiment, the answer contains the

1 minimum length of the time interval that the request processor must wait
 2 before sending another segment of the same packet. The number of lines
 3 **160** into RP/OC_N **154** is usually greater than the number of segments that are
 4 given permission to enter the RP/OC_N . In this way, the segments that have
 5 been scheduled to exit the RS/DS switch are able to pass through the RS/DS
 6 switch into the output controllers, while the segments that are also requests
 7 have a path into RP/OC_N as well. In case the number of request segments
 8 plus the number of scheduled segments exceeds the number of lines **160**
 9 from RS/DS switch **156** into output controller **154**, then the excess packets
 10 are buffered internally in switch RS/DS **156** and can enter the target RP/OC
 11 at the next cycle.

12 In case a packet is not able to exit the switch immediately because all
 13 of the output lines are blocked, there is a procedure to keep the segments of
 14 a data packet from getting out of order. This procedure also keeps the
 15 RS/DS from becoming overloaded. For a packet segment S_M traveling from
 16 an input controller IC_P to an output controller section of RP/OC_K , the
 17 following procedure is followed. When the packet segment S_M enters
 18 RP/OC_K , then RP/OC_K sends an acknowledgement packet (not illustrated)
 19 through answer switch AS **108** to IC_P **150**. Only after IC_P has received the
 20 acknowledgement packet will it send the next segment, S_{M+1} . Since the
 21 answer switch only sends acknowledgements for packet segments that
 22 successfully pass through the RS/DS switch into an output controller, the
 23 segments of a packet cannot get out of sequence. An alternate scheme is to
 24 include a segment number field in the segment packet, which the output
 25 controller uses to properly assemble the segments into a valid packet for
 26 transmission downstream.

1 The acknowledgement from RP/OC_K to IC_P is sent in the form of an
2 answer packet illustrated in **FIG. 2E**. Since the payload of this packet is
3 short relative to the length of the segment packet, the system can be
4 designed so that an input controller sending the segment S_M to RP/OC_K will
5 typically receive an answer before it has finished inserting the entire
6 segment S_M into switch RS/DS. In this way, in case the answer is
7 affirmative, the input port processor can advantageously begin the
8 transmission of segment S_{M+1} immediately following the transmission of
9 segment S_M.

10 An input controller receives no more than one answer for each request
11 it makes. Therefore, the number of answers per unit time received by an
12 input controller is not greater than the number of requests per unit time sent
13 from the same input controller. Advantageously, an answer switch
14 employing this procedure cannot become overloaded since all answers sent
15 to a given input controller are in response to requests previously sent by that
16 controller.

17 Referring to **FIG. 1A**, in an alternate embodiment not illustrated,
18 request switch **104** and answer switch **108** are implemented as a single
19 component, which handles both requests and answers. These two functions
20 are performed by a single MLML switch fabric alternately handling requests
21 and answers in a time-sharing fashion. This switch carries out the function
22 of request switch **104** at one time, and the function of answer switch **108** at
23 the next. An MLML switch fabric that is suitable for implementing request
24 switch **104** is generally suitable for the combined function discussed here.
25 The function of request processor **106** is handled by an RP/OC processor
26 **154**, such as those described for **FIGs. 1E** and **1F**. The operation of the
27 system in this embodiment is logically equivalent to the controlled switch

1 system **100**. This embodiment advantageously reduces the amount of
2 circuitry needed to implement control system **120**.

3 SINGLE SWITCH EMBODIMENT

4 **FIG. 1F** illustrates an embodiment of the invention wherein switch
5 RADS **158** carries and switches all of the packets for the request switch, the
6 answer switch and the data switch. In this embodiment, it is useful to use
7 the multiple-length-packet switch described later for **FIGs. 12B** and **14**. The
8 operation of the system in this embodiment is logically equivalent to the
9 combined data switch and request switch embodiment described for **FIG.**
10 **1E**. This embodiment advantageously reduces the amount of circuitry
11 needed to implement control system **120** and data switch system **130**.

12 The control systems discussed above can employ two types of flow
13 control schemes. The first scheme is a request-answer method, where data is
14 sent by input controller **150** only after an affirmative answer is received
15 from request processor **106**, or RP/OC processor **154**. This method can also
16 be used with the systems illustrated in **FIGs. 1A** and **1E**. In these systems, a
17 specific request packet is generated and transmitted to the request processor,
18 which generates an answer and sends it back to the input controller. The
19 input controller always waits until it receives an affirmative answer from the
20 RP/OC processor before sending the next segment or remaining segments.
21 In the system illustrated in **FIG. 1E** the first data segment can be treated as a
22 combined request packet and data segment, where the request pertains to the
23 next segment, or to all the remaining segments.

24 The second scheme is a "send-until-stopped" method where the input
25 controller sends data segments continuously unless the RP/OC processor
26 sends a halt-transmission or pause-transmission packet back to the input

1 controller. A distinct request packet is not used as the segment itself implies
 2 a request. This method can be used with the systems illustrated in **FIGs. 1E**
 3 and **1F**. If the input controller does not receive a halt or pause signal, it
 4 continues transmitting segments and packets. Otherwise, upon receiving a
 5 halt signal it waits until it receives a resume-transmission packet from the
 6 RP/OC processor; or upon receiving a pause signal it waits for the number of
 7 time periods indicated in the pause-transmission packet and then resumes
 8 transmission. In this manner traffic moves promptly from input to output,
 9 and impending congestion at an output is immediately regulated, desirably
 10 preventing an overload condition at the output port. This "send-until-
 11 stopped" embodiment is especially suitable for an Ethernet switch.

12 A massively parallel computer could be constructed so that the
 13 processors could communicate via a large single-switch network. One
 14 skilled in the art could use the techniques of the present invention to
 15 construct a software program in which the computer network served as a
 16 request switch, an answer switch and a data switch. In this way, the
 17 techniques described in this patent can be employed in software.

18 In this single switch embodiment as well as in other embodiments,
 19 there are a number of answers possible. When a request to send a packet is
 20 received, the answers include but are not limited to: 1) send the present
 21 segment and continue sending segments until the entire packet has been sent;
 22 2) send the present segment but make a request later to send additional
 23 segments; 3) at some unspecified time in the future, re-submit a request to
 24 send the present segment; 4) at a prescribed time in the future, resubmit a
 25 request to send the present packet; 5) discard the present segment; 6) send
 26 the present segment now and send the next segment at a prescribed time in

1 the future. One of ordinary skill in the art will find other answers that fit
2 various system requirements.

3 MULTICASTING USING LARGE MLML SWITCHES

4 Multicasting refers to the sending of a packet from one input port to a
5 plural number of output ports. In many of the electronic embodiments of the
6 switches disclosed in the present patent and in the patents incorporated by
7 reference, the logic at a node is very simple, not requiring many gates.
8 Minimal chip real estate is used for logic as compared to the amount of I/O
9 connections available. Consequently, the size of the switch is limited by the
10 number of pins on the chip rather than the amount of logic. Accordingly,
11 there is ample room to put a large number of nodes on a chip. Since the
12 lines 122 carrying data from the request processors to the request switch are
13 on the chip, the bandwidth across these lines can be much greater than the
14 bandwidth through the lines 134 into the input pins of the chip. Moreover, it
15 is possible to make the request switch large enough to handle this
16 bandwidth. In a system where the number of rows in the top level of the
17 MLML network is N times the number of input controllers, it is possible to
18 multicast a single packet to as many as N output controllers. Multicasting to
19 K output controllers (where $K \leq N$) can be accomplished by having the input
20 controllers first submit K requests to the request processor, with each
21 submitted request having a separate output port address. The request
22 processor then returns L approvals ($L \leq K$) to the input controller. The input
23 controller then sends L separate packets through the data switch with the L
24 packets each having the same payload but a different output port address. In
25 order to multicast to more than N outputs, it is necessary to repeat the above
26 cycle a sufficient number of times. In order to accomplish this type of

1 multicasting, the input controllers must have access to stored multicast
2 address sets. The necessary changes to the basic system necessary to
3 implement this type of multicasting will be obvious to one skilled in the art.

4 SPECIAL MULTICASTING HARDWARE

5 **FIGs. 4A, 4B and 4C** show another embodiment of system **100** that
6 supports multicasting. Request controller **120** shown in **FIG. 1A** has been
7 replaced with multicasting request controller **420**, and data switch **130** has
8 been replaced with multicasting data switch **440**. The multicasting
9 techniques employed here are based on those taught in Invention #5. A
10 multicast packet is sent to a plurality of output ports, that taken together
11 form a multicast set. There is a fixed upper limit on the number of members
12 in the multicast set. If the limit is L and if there are more than L members in
13 the actual set, then a plurality of multicast sets is used. An output port may
14 be a member of more than one multicast set.

15 Multicast SEND requests are accomplished via indirect addressing.
16 Logic units LU come in pairs, **432** and **452**, one in the request controller **420**
17 and one in the data switch **440**. Each pair of logic units share a unique
18 logical output port address OPA **204**, which is distinct from any physical
19 output port address. The logical address represents a plural number of
20 physical output addresses. Each logic unit of the pair contains a storage
21 ring, and each of these storage rings is loaded with an identical set of
22 physical output port addresses. The storage ring contains the list of
23 addresses, in effect forming a table of addresses where the table is
24 referenced by its special address. By employing this tabular output-port
25 address scheme, multicast switches, RMC_T **430** and DMC_T **450**, efficiently
26 process all multicast requests. Request packets and data packets are

1 replicated by the logic units **432** and **452**, in concert with their respective
 2 storage rings **436** and **456**. Accordingly, a single request packet sent to a
 3 multicast address is received by the appropriate logic unit **432** or **452**, which
 4 in turn, replicates the packet once for each item in the table contained in its
 5 storage ring. Each replicated packet has a new output address taken from
 6 the table, and is forwarded to a request processor **106** or output controller
 7 **110**. Non-multicast requests never enter the multicast switches RMC_T **430**,
 8 but are instead directed to bottom levels of switch RS_B **426**. Similarly, non-
 9 multicast data packets never enter the multicast data switches DMC_T **450**,
 10 but are instead directed to bottom levels of switch DS_B **444**.

11 **FIGs. 2G, 2H, 2I, 2J, 2K and 2L** show additional packet and field
 12 modifications for supporting multicasting. **Table 2** is an overview of the
 13 contents of these fields.

14

MAM	A bitmask indicating approval for a single address requested by a multicast send packet.
MF	A one-bit field that indicates a multicast packet.
MLC	A two-bit field that tracks the status of the two LOADs needed to update a set of multicast addresses in storage rings 436 and 456 .
MLF	A one-bit field indicating that a packet wants to update a set of multicast addresses stored in the switches.
MRM	A bitmask that keeps track of pending approvals needed to complete a multicast SEND request.
MSM	A bitmask that that keeps track of approvals for a multicast SEND request which have not yet been processed by the multicast data switch.
PLBA	Address in the multicast LOAD buffer where LOAD packets are stored. Used instead of the packet buffer address PBA when a multicast load is requested.

Table 2

LOADING MULTICAST ADDRESS SETS

Loading of storage rings **436** and **456** is accomplished using a multicast packet **205**, given in **FIG. 2G**, whose format is based on that of the packet **200**. A system processor **140** generates the LOAD requests. When the packet arrives at an input controller IC **150**, the input controller processor **160** examines the output port address OPA **204** and notes by the address that that a multicast packet has arrived. If the multicast load flag

MLF **203** is on, the packet is a multicast load and the set of addresses to be loaded resides in the PAY field **208**. In one embodiment, the logical output port address that is given has been previously supplied to the requestor. In other embodiments, the logical output port address is a dummy address that triggers the controller to select the logical output port address for a pair of available logic units; this OPA will be returned to the requestor for use when sending the corresponding multicast data packets. In either case, the input controller processor then generates and stores a packet entry **225** in its multicast load buffer **418** and creates a multicast buffer KEY entry **215** in its KEYs buffer **166**. The buffer KEY **215** contains a two-bit multicast load counter MLC **213** that is turned on to indicate that a LOAD request is ready for processing. The multicast load buffer address PLBA **211** contains the address in the multicast load buffer where the multicast load packet is stored. During a request cycle, the input controller processor sends the multicast load packet to the request controller **420** to load the storage ring in the logic unit at address OPA **204**, and then turns off the first bit of MLC **213** to indicate that this LOAD has been done. Similarly, the input controller processor selects a data cycle in which it sends the same multicast load packet to the data controller **440**, and the second bit of MLC **213** is turned off. When both bits of MLC **213** have been turned off, the input controller processor can remove all information for this request from its KEYs buffer and multicast load buffer since its part in the load request has been completed. Processing of the multicast load packet is the same at both the request controller **420** and the data controller **440**. Each controller uses the output port address to send the packet through its MC_T switch to its appropriate logic unit LU **432** or LU **452**. Since the multicast load flag MLF **203** is on, each logic unit notes that it has been asked to update the addresses

in its storage ring by using the information in the packet payload PAY **208**. This update method synchronizes the sets of addresses in corresponding storage ring pairs.

MULTICASTING DATA PACKETS

Multicast packets are distinguished from non-multicast packets by their output port addresses OPA **204**. Multicast packets not having the multicast load flag MLF **203** turned on are called multicast send packets. When the input controller processor **160** receives a packet **205** and determines from the output port address and multicast load flag that it is a multicast send packet, the processor makes the appropriate entries in its packet input buffer **162**, request buffer **164** and KEYs buffer **166**. Two special fields in the multicast buffer KEY **215** are used for SEND requests. The multicast request mask MRM **217** keeps track of which addresses are to be selected from those in the target storage ring. This mask is initially set to select all addresses in the ring (all ones). The multicast send mask MSM **219** keeps track of which requested addresses have been approved by the request processors, RP **106**. This mask is initially set to all zeros, indicating that no approvals have yet been given.

When the input controller processor examines its KEYs buffer and selects a multicast send entry to submit to the request controller **420**, the buffer key's current multicast request mask is copied into the request packet **245** and the resulting packet is sent to the request processor. The request switch RS **424** uses the output port address to send the packet to the multicast switch RMC_T, which routes the packet on to the logic unit LU **432** designated by OPA **204**. The logic unit determines from MLF **203** that it is not a load request, and uses the multicast request mask MRM **217** to decide

which of the addresses in its storage ring to use in multicasting. For each selected address, the logic unit duplicates the request packet **245** making the following changes. First, the logical output port address OPA **204** is replaced with a physical port address from selected ring data. Second, the multicast flag MLF **203** is turned on so that that the request processors know that this is a multicast packet. Third, the multicast request mask is replaced by a multicast answer mask MAM **251**, which identifies the position of the address from the storage ring that was loaded into the output port address. For example, the packet created for the third address in the storage ring has the value 1 in the third mask bit and zeros elsewhere. The logic unit sends each of the generated packets to the switch RMC_B that uses the physical output port address to send the packet to the appropriate request processor, RP **106**.

Each request processor examines its set of request packets and decides which ones to approve and then generates a multicast answer packet **255** for each approval. For multicast approvals, the request processor includes the multicast answer mask MAM **251**. The request processor sends these answer packets to the answer switch AS **108**, which uses IPA **230** to route each packet back to its originating input control unit. The input controller processor uses the answer packet to update buffer KEY data. For multicast SEND requests this includes adding the output port approved in the multicast answer mask to the multicast send mask and removing it from the multicast request mask. Thus, the multicast request mask keeps track of addresses that have not yet received approval, and the multicast send mask keeps track of those that have been approved and are ready to send to the data controller **440**.

During the SEND cycle, approved multicast packets are sent to the data controller as multicast segment packets **265** that include the multicast send mask MSM **219**. The output port address is used by the data switches DS **442** and MC_T **430** to route the packet to the designated logic unit. The logic unit creates a set of multicast segment packets, each identical to the original packet, but having a physical output port address supplied by the logic unit according to the information on the multicast send mask. The modified multicast segment packets then pass through the multicast switch MC_B, which sends them to the proper output controller **110**.

The output controller processor **170** reassembles the segment packets by using the packet identifiers, KA **228** and IPA **230**, and the NS **226** field. Reassembled segment packets are placed in the packet output buffer **172** for sending to LC **102**, thus completing the SEND cycle. Non-multicasting packets are processed in a similar manner, except that they bypass the multicast switch **448**. Instead, the data switch **442** routes the packet through switch DS **444** based on the packet's physical output port address OPA **204**.

MULTICAST BUS SWITCH

FIGs. 5A and 5B are diagrams showing an alternate method for implementing and supporting multicasting using an on-chip bus structure. **FIG. 5A** is a diagram showing a plurality of request processors **516** interconnected by means of a multicast request bus switch **510**. **FIG. 5B** is a diagram showing a plurality of output processors **546** interconnected by means of a data-packet-carrying multicast bus switch **540**.

A multicast packet is sent to a plurality of output ports, which taken together form a multicast set. Bus **510** allows connections to be sent to specific request processors. The multicast bus functions like an M-by-N

crossbar switch, where M and N need not be equal, and where the links, **514** and **544**. One connector **512** in the bus represents one multicast set. Each request processor has the capability of forming an I/O link **514** with zero or more connectors **512**. These links are set up prior to the use of the buses. A given request processor **516** only links to connectors **512** that represent the multicast set or sets to which it belongs, and is not connected to other connectors in the bus. The output port processors **546** are similarly linked to zero or more data-carrying connectors **542** of output multicast bus **540**. Those output port processors that are members of the same set have an I/O link **544** to a connector **542** on the bus representing that set. These connection links, **514** and **544**, are dynamically configurable. Accordingly, special MC LOAD messages add, change and remove output ports as members of a given multicast set.

One request processor is specified as the representative (REP processor) of a given multicast set. An input port processor sends a multicast request only to the REP processor **518** of the set. **FIG. 6C** illustrates a multicast timing scheme where multicast requests are made only at designated time periods, MCRC **650**. If an input controller **150** has one or more multicast request in its buffer, it waits for a multicast request cycle **650** to send its requests to a REP processor. A REP processor that receives a multicast request informs the other members of the set by sending a signal on the shared bus connector **512**. This signal is received by all other request processors linked to the connector. If a REP processor receives two or more multicast requests at the same time, it uses priority information in the requests to decide which requests are placed on the bus.

After the REP processor has selected one or more requests to put on the bus, it uses connector **512** to interrogate other member of the set before

1 sending an answer packet back to the winning input controller. A request
 2 processor may be a member of one or more multicast sets, and may receive
 3 notification of two or more multicast requests at one time. Alternately
 4 stated, a request processor that is a member of more than one multicast set
 5 may detect that a plurality of multicast bus connections **514** are active at one
 6 time. In such a case, it may accept one or more requests. Each request
 7 processor uses the same bus connector to inform the REP processor that it
 8 will accept (or refuse) the request. This information is transmitted over
 9 connector **512** from each request processor to the REP processor by using a
 10 time-sharing scheme. Each request processor has a particular time slot when
 11 it signals its acceptance or refusal. Accordingly, the REP processor receives
 12 responses from all members in bit-serial fashion, one bit per member of the
 13 set. In an alternate embodiment, non-REP processors inform the REP
 14 processor ahead of time that they will be busy.

15 The REP processor then builds a multicast bit-mask that indicates
 16 which members of the multicast set accept the request; the value 1 indicates
 17 acceptance, the value 0 indicates refusal, and the position in the bitmask
 18 indicates which member. The reply from the REP processor to the input
 19 controller includes this bitmask and is sent to the requesting input controller
 20 by means of the answer switch. The REP processor also sends a rejection
 21 answer packet back to an input controller in case the bit-mask contains all
 22 zeros. A denied multicast request may be reattempted at a subsequent
 23 multicast cycle. In an alternative embodiment, each output port keeps a
 24 special buffer area for each multicast set for which it is a member. At a
 25 prescribed time, an output port sends a status to each of the REP processors
 26 corresponding to its multicast sets. This process continues during data
 27 sending cycles. In this fashion, the Rep knows in advance which output

1 ports are able to receive multicast packets and therefore is able to respond to
2 multicast requests immediately without sending requests to all of its
3 members.

4 During the multicast data cycle, an input controller with an acceptance
5 multicast response inserts the multicast bitmask into the data packet header.
6 The input controller then sends the data packet to the output port processor
7 that represents the multicast set at the output. Recall that the output port
8 processors are connected to multicast output bus **540**, analogous to the
9 means whereby request processors are connected to multicast bus **510**. The
10 output port processor REP that receives the packet header transmits the
11 multicast bitmask on the output bus connector. An output port processor
12 looks for 0 or 1 at a time corresponding to its position in the set. If 1 is
13 detected, then that output port processor is selected for output. After
14 transmitting the multicast bitmask, the REP output port processor
15 immediately places the data packet on the same connector. The selected
16 output port processors simply copy the payload to the output connection,
17 desirably accomplishing the multicast operation. In alternate embodiments,
18 a single bus connector, **512** and **542**, that represents a given multicast set
19 may be implemented by a plurality of connectors, desirably reducing the
20 amount of time it takes to transmit the bit-mask. In another embodiment,
21 where the multicast packet is sent only in case all of the outputs on a bus can
22 accept a packet, a 0 indicates an acceptance and a 1 indicates a rejection. All
23 processors respond at the same time and if a single one is received, then the
24 request is denied.

25 A request processor that receives two or more multicast requests may
26 accept one or more requests, which are indicated by 1 in the bitmask
27 received back by the requesting input controller. A request processor that

rejects a request is indicated by 0 in the bit-mask. If an input controller does not get all 1's (indicating 100% acceptance) for all members of the set then it can make another attempt at a subsequent multicast cycle. In this case, the request has a bitmask in the header that is used to indicate which members of the set should respond to or ignore the request. In one embodiment, multicast packets are always sent from the output processor immediately when they are received. In another embodiment, the output port can treat the multicast packets just like other packets and can be stored in the output port buffer to be sent at a later time.

An overload condition can potentially occur when upstream devices frequently send multicast packets, or when two or more upstream sources send a lot of traffic to one output port. Recall that all packets that exit an output port of the data switch must have been approved by the respective request processor. If a given request processor receives too many requests, whether as a result of multicast requests or because many input sources want to send to the output port or otherwise, the request processor accepts only as many as can be sent through the output port. Accordingly, an overload at an output port cannot occur when using the control system disclosed here.

Referring also to **FIG. 1D** an input controller that is denied permission to send packets through the data switch can try later. Importantly, it can discard packets in its buffer when an impending overload occurs. The input controller has sufficient information about which packets are not being accepted for which output ports such that it may evaluate the situation and determine the type and cause of the overload. It can then inform the system processor **140** of this situation by sending it a packet through the data switch. Recall that the system processor has a plurality of I/O connections to the control system **120** and to the data switch **130**. The

1 system processor can process packets from one or more input controllers at
 2 one time. System processor **140** can then generate and send appropriate
 3 packets to upstream devices to inform them of the overload condition so that
 4 the problem may be fixed at the source. The system processor can also
 5 inform a given input port processor to ignore and discard certain packets that
 6 it may have in its buffer and may receive in the future. Importantly, the
 7 scalable switching system disclosed here is immune to overloading,
 8 regardless of cause, and is therefore regarded as congestion-free.

9 The multicast packets can be sent through the data switch at a special
 10 time or at the same time with other data. In one embodiment, a special bit
 11 informs a REP output port processor that the packet is to be multicast to all
 12 of the members of the bus or to those members in some bit-mask. In the
 13 later case, a special set up cycle sets the switches to the members selected by
 14 the bit-mask. In another embodiment, packets are sent through the special
 15 multicast hardware only if all members of the bus are to receive the packet.
 16 It is possible that the number of multicast sets is greater than the number of
 17 output ports. In other embodiments, there are a plural number of multicast
 18 sets with each output port being a member of only one multicast set. Three
 19 methods of multicasting have been presented. They include:

- 20 1. the type of multicasting that requires no special hardware in
 21 which a single packet arriving into the input controller causes a
 22 plurality of requests to be sent to the request switch and a
 23 plurality of packets to be sent to the data switch,
- 24 2. a type of multicasting using the rotating FIFO structure taught
 25 in Invention #5, and
- 26 3. a type of multicasting requiring a multicast bus.

A given system using multicasting can employ one, two, or all three of these schemes.

SYSTEM TIMING

Referring to **FIG. 1A**, an arriving packet enters system **100** through input line **126** on line card **102**. The line card parses the packet header and other fields to determine where to send it and to determine priority and quality of service. This information, along with the packet, is sent over path **134** to connected input controller **150**. The input controller uses this information to generate a request packet **240** that it sends into control system **120**. In the control system, request switch **104** transmits the request packet to a request processor **106** that controls all traffic sent to a given output port. In the general case, one request processor **106** represents one output port **110**, and controls all traffic such that no packet is ever sent to a system output port **128** without having been approved by the corresponding request processor. In some embodiments the request processor **106** is physically connected to the output controller **110**, as shown in **FIGs. 1E** and **1F**. The request processor receives the packet; it may receive requests from other input controllers that also have data packets wanting to be sent to the same output port. The request processor ranks the requests based on priority information in each packet and may accept one or more requests while denying other requests. It immediately generates one or more answer packets **250** that are sent through answer switch **108** to inform the input controllers of accepted “winning” and rejected “loosing” packets. An input controller with an accepted data packet sends the data packet into data switch **130** that transmits it to an output controller **110**. The output controller removes any internally used fields and sends it to the line card

over path 132. The line card converts the packet into a format suitable for physical transmission downstream 128. A request processor that rejects one or more requests additionally may send answer packets indicating rejections to input controllers to provide them with information that they use to estimate the likelihood of acceptance of the packet at a later cycle.

Referring also to FIG. 6A, the timing of request and answer processing is overlapped with transmission of data packets through the data switch, which is also overlapped with packet reception and parsing performed by the line card in conjunction with the input controller. An arriving packet K 602 is first processed by the line card that examines the header and other relevant packet fields 606 to determine the packet's output port address 204 and QOS information. A new packet arrives at time T_A at the line card. At time T_R the line card has received and processed sufficient packet information such that the input controller can begin its request cycle. The input controller generates request packet 240. Time period T_{RQ} 610 is the time that the system uses to generate and process requests, and to receive and answer at the winning input controller. Time period T_{DC} 620 is the amount of time that the data switch 130 uses to transmit a packet from its input port 116 to output port 118. In one embodiment, T_{DC} is a longer period than T_{RQ} .

In the example illustrated in FIG. 6A, a packet K 602 is received by the line card at time T_A . The input controller generates request packet 240 that is handled by the control system during time period T_{RQ} . During this time period a previously arriving packet J 620 moves through the data switch. Also during time period T_{RQ} , another packet L 622 is arriving at the line card. Importantly, because a request processor sees all requests for its output port and accepts no more than could cause congestion, the data switch

is never overloaded or congested. Input controllers are provided with necessary and sufficient information to determine what to do next with packets in its buffers. Packets that must be discarded are equitably chosen based on all relevant information in their header. Request switch **104**, answer switch **108**, and data switch **130** are scalable, wormholing MLML interconnects of the types taught in Inventions #1, #2 and #3. Accordingly, requests are processed in overlapped fashion with data packet switching, such that scalable, global control of the system is advantageously performed in a manner that permits data packets to move through system without delay.

FIG. 6B is a timing diagram that shows in more detail the steps of overlapped processing of an embodiment that also supports multiple, request sub-cycles. The following list refers to numbered lines **630** of the diagram:

1. The input controller, IC **150**, has received sufficient information from the line card to construct a request packet **240**. The input controller may have other packets in its input buffer and may select one or more of them as its top priority requests. Sending the first request packet or packets into the request switch at time T_R marks the beginning of the request cycle. After time T_R , if there is at least one more packet in its buffer for which there was no first round request and in case one or more of the first round requests is rejected, the input controller immediately prepares second priority request packets (not shown) for use in a second (or third) request sub-cycle.
2. Request switch **104** receives the first bits of the request packet at time T_R , and sends the packet to the target request processor specified in OPA field **204** of the request.
3. In this example, the request processor receives up to three requests that arrive serially starting at time T_3 .

4. When the third request has arrived at time T_4 , the request processor ranks the requests based on priority information in the packets, and may select one or more requests to accept. Each request packet contains the address of the requesting input controller. The address of the requesting input controller is used as the target address of the answer packet.
5. Answer switch **108** transmits using the IPA address to send the acceptance packets to the input controller making the requests.
6. The input controller receives acceptance notification at time T_6 and sends the data packet associated with the acceptance packet into the data switch at the start of the next data cycle **640**. Data packets from the input controllers enter the data switch at time T_D .
7. The request processor generates rejection answer packets **250** and sends them through the answer switch to the input controllers making the rejected requests.
8. When the first rejection packet is generated, it is sent into the answer switch **108** followed by other rejection packets. The final rejection packet is received by the input controller at time T_8 . This marks the completion of the request cycle, or the first sub-cycle in embodiments employing multiple request sub-cycles.
9. Request cycle **160** starts at time T_R and ends at time T_8 for duration T_{RQ} . In an embodiment that supports request sub-cycles, request cycle **610** is considered to be the first sub-cycle. The second sub-cycle **612** begins at time T_8 after all of the input controllers have been informed of the accepted and rejected requests. During the time between T_3 and T_8 , an input controller with packets for which there was no request on the first cycle, builds request packets for the second sub-cycle. These

requests are sent at time T_8 . When more than one sub-cycle is used, the data packets are sent into the data switch at the completion of the last sub-cycle (not shown).

This overlapped processing method advantageously permits the control system to keep pace with the data switch. This overlapped processing method advantageously permits the control system to keep pace with the data switch.

FIG. 6C is a timing diagram of an embodiment of a control system that supports a special multicast processing cycle. In this embodiment multicast requests are not permitted at non-multicast (normal) request cycles, **RC 610**. An input controller that has a packet for multicast waits until the multicast request cycle, **MCRC 650**, to send its request. Accordingly, multicast requests do not compete with normal requests, advantageously increasing the likelihood that all targets ports of the multicast will be available. The ratio of normal to multicast cycles and their timing are dynamically controlled by the system processor **140**.

FIG. 6D is a timing diagram of an embodiment of a control system that supports time-slot reservation scheduling discussed with **FIGs. 3A, 3B** and **3C**. This embodiment exploits the fact that, on average, a data packet is subdivided into a significant number of segments and only one request is made for all segments of a packet. A single time-slot reservation request packet **310** is sent and answer packet **320** is received during one time-slot request cycle, **TSRC 660**. After the answer is received, the plurality of segments are sent during shorter, time-slot data cycles, **TSDC 662**, at the rate of one segment per TSDC cycle. In an example, assume that the average data packet is broken into 10 segments. This means that for every 10 segments sent into the data switch, the system has to perform only one

TSRC cycle. Thus, request cycle **660** could be 10 times longer than the data cycle **662**, and control system **120** could still handle all incoming traffic. In practice, a ratio less than the average should be used to accommodate situations where an input port receives a burst of short packets.

POWER SAVING SCHEMES

There are two components in the MLML switch fabric that serially transmit packet bits. These are: 1) Control cells and 2) FIFO buffers at each row of the switch fabric. Referring to FIGS 8 and 13A, a clock signal **1300** causes data bits move in bucket-brigade fashion through these components. In a preferred embodiment of the MLML switch fabric, simulations indicate that only 10% to 20% of these components have a packet transiting through them at a given time; the remainder are empty. But even when there is no packet present (all zeros) the shift registers consume power. In a power-saving embodiment the clock signal is appropriately turned off when no packet is present.

In a first power-saving scheme, the clock driving a given cell is turned off as soon as the cell determines that no packet has entered it. This determination takes only a single clock cycle for a given control cell. At the next packet arrival time **1302** the clock is turned on again, and the process repeats. In a second power-saving scheme, the cell that sends a packet to the FIFO on its row determines whether or not a packet will enter the FIFO. Accordingly, this cell turns the FIFO's clock on or off.

If no cell in an entire control array **810** is receiving a packet, then no packets can enter any cell or FIFO to the right of the control array on the same level. In a third power-saving scheme, when no cell in a control array

sends a packet to its right, the clocks are turned off for all cells and FIFOs on the same level to the right of this control array.

CONFIGURABLE OUTPUT CONNECTIONS

The traffic rate at an output port can vary over time, and some output ports can experience a higher rate than others. **FIG. 7** is a diagram of the bottom level of an MLML data switch of the type taught in Inventions #2 and #3 showing how configurable connections are made to physical output ports **118**. A node **710** at the bottom level of the switch has a settable connection **702** to an output port **118** of the switch chip. Node A on row address 0 connects by means of link **702** to one output port **118**; nodes B, C and D are on row 1, **704** and have the same output address. At three columns, nodes B, C and D connect to three different physical output ports **706**. Similarly, output addresses 5 and 6 each connect to two output ports. Accordingly, output addresses 1, 5 and 6 have higher bandwidth capacity at the data switch output.

TRUNKING

Trunking refers to the aggregation of a plurality of output ports that are connected to a common downstream connection. At the data switch, output ports connected to one trunk are treated as a single address, or block of addresses, within the data switch. Different trunks can have different numbers of output port connections. **FIG. 8** is a diagram of the bottom levels of an MLML data switch of the type taught Inventions #2 and #3 that has been modified to support trunking. A node is configured by a special message sent by the system processor **140** so that it either reads or ignores header address bits. A node **802**, indicated by “x”, ignores packet header

bits (address bits) and routes the packet down to the next level. Nodes at the same level that reach the same trunk are shown inside a dashed box **804**. In the illustration, output addresses 0, 1, 2 and 3 connect to the same trunk, TR0 **806**. A data packet sent to any of these addresses will exit the data switch at any of the four output ports **118** of TR0. Alternately stated, a data packet with output address 0, 1, 2 or 3 will exit the switch at any of the four ports of trunk TR0. Statistically, any output port **118** of trunk TR0 **806** is equally likely to be used, regardless of the packet's address: 0, 1, 2 or 3. This property advantageously smoothes out traffic flowing out from among the plurality of output connections **118**. Similarly, packets sent to address 6 or 7 are sent out trunk TR6 **808**.

PARALLELIZATION FOR HIGH-SPEED I/O AND MORE PORTS

When segmentation and reassembly (SAR) is utilized, the data packets sent through the switch contain segments rather than full packets. In one embodiment of the system illustrated in **FIG. 1A** employing the timing scheme illustrated in **FIG. 6D**, the request processors can, at one time, give permission for all of the segments of a packet to be sent to their target output controller. The input controller makes a single request that indicates how many segments are in the complete packet. The request processor uses this information in ranking requests; when a multi-segment request has been granted, the request processor does not allow any subsequent request until such time that all segments have been sent. The input controllers, the request switch, request processors, and the answer switch desirably have a reduced workload. In such an embodiment the data switch is kept busy while the request processor is relatively idle. In this embodiment, request

cycle **660** can be of longer duration than data (segment) switch cycle **662**, advantageously relaxing design and timing constraints for the control system **120**.

In another embodiment the rate through the data switch is increased without increasing the capacity of the request processor. This can be achieved by having a single controller **120** managing the data going into multiple data switches, as illustrated by the switch and control system **900** of **FIG. 9**. In one embodiment of this design, in a given time period, each of the input controllers **990** is capable of sending a packet to each of the data switches in the stack of data switches **930**. In another embodiment the input controller can decide to send different segments of the same packet to each of the data switches, or it can decide to send segments from different packets to the data switches. In other embodiments, at a given time step, different segments of the same packet are sent to different data switches. In yet another embodiment, one segment is sent to the entire stack of data switches in bit-parallel fashion, reducing the amount of time for the segment to wormhole through the data switch by an amount proportional to the number of switch chips in the stack.

In **FIG. 9**, the design allows for a plural number of data switches that are managed by request controller **120** with a single request switch and a single answer switch. In other designs, the request controller contains a plural number of request switches **104** and a plural number of answer switches **108**. In yet other designs, there are a multiple number of request switches and a multiple number of answer switches as well as a multiple number of data switches. In the last case, the number of data switches could be equal to the number of request control units or the number of request

1 processors could be either greater than or less than the number of data
2 switches.

3 In the general case, there are P request processors that handle only
4 multicast requests, Q data switches for handling only multicast packets, R
5 request processors for handling direct requests, and S data switches for
6 handling direct addressed data switching.

7 A way to advantageously employ multiple copies of request switches
8 is to have each request switch receive data on J lines with one line arriving
9 from each of the J input controller processors. In this embodiment, one of
10 the duties of the input processors is to even out the load to the request
11 switches. The request processors use a similar scheme in sending data to the
12 data switch.

13 Referring to **FIG. 1D**, a system processor **140** is configured to send
14 data to and receive data from the line cards, the input processors, and the
15 request processors, and to communicate with external devices outside of the
16 system, such as an administration and management system. Data switch I/O
17 ports **142** and **144**, and control system I/O ports, **146** and **148**, are reserved
18 for use by the system processor. The system processor can use the data
19 received from the input processors and from the request processors to inform
20 a global management system of local conditions, and to respond to the
21 requests of the global management system. The algorithms and methods
22 that the request processors use to make their decisions can be based on a
23 table lookup procedure, or on a simple ranking of requests by a single-value
24 priority field. Based on information from within and without the system, the
25 system processor can alter the algorithm used by the request processors, for
26 example by altering their lookup tables. An IC WRITE message (not
27 shown) is sent on path **142** into the data switch to an output controller **110**

1 that transmits over path **152** to the associated input controller **150**.
 2 Similarly, an IC READ message is sent to an input controller, which
 3 responds by sending its reply through the data switch to the port address **144**
 4 of the system processor. An RP WRITE message (not shown) is used to
 5 send information to a request processor on path **146** using the request switch
 6 **104**. An RP READ message is similarly used to interrogate a request
 7 processor, which sends its reply through answer switch **108** to the system
 8 processor on path **148**.

9 **FIG. 10A** illustrates a system **1000** where yet another degree of
 10 parallelism is achieved. Multiple copies of the entire switch, **100** or **900**,
 11 including its control system and data switch, are used as modules to
 12 construct larger systems. Each of the copies is referred to as a layer **1004**;
 13 there can be any number of layers. In one embodiment, K copies of the
 14 switch and control system **100** are used to construct a large system. A layer
 15 may be a large optical system, a layer may consist of a system on a board, or
 16 a layer may consist of a system in one rack, or of many racks. It is
 17 convenient in what follows to think of a layer as consisting of a system on a
 18 board. In this way, a small system can consist of only one board (one layer)
 19 whereas larger systems consist of multiple boards.

20 For the simplest layer, as depicted in **FIG. 1A**, a list of the
 21 components on a layer m follows:

- 22 • One data switch DS_m
- 23 • One request switch RS_m
- 24 • One request processor, RC_m
- 25 • One answer switch AS_m
- 26 • J request processors, $RP_{0,m}, RP_{1,m}, \dots RP_{J-1,m}$

- J input controllers, $IC_{0,m}, IC_{1,m}, \dots IC_{J-1,m}$
- J output controllers, $OC_{0,m}, OC_{1,m}, \dots OC_{J-1,m}$

A system with the above components on each of K layers has the following “parts count:” K data switches, K request switches, K answer switches, J-K input controllers, J-K output controllers and J-K request processors.

In one embodiment, there are J line cards $LC_0, LC_1, \dots LC_{J-1}$, with each line card **1002** sending data to every layer. In this embodiment, the line card LC_n feeds the input controllers $IC_{n,0}, IC_{n,1}, \dots, IC_{n,k-1}$. In an example where an external input line **1020** carries wave division multiplexed (WDM) optical data with K channels, the data can be demultiplexed and converted into electronic signals by optical-to-electronic (O/E) units. Each line card receives K electronic signals. In another embodiment, there are K electronic lines **1022** into each line card. Some of the data input lines **126** are more heavily loaded than others. In order to balance the load, the K signals entering a line card from a given input line can advantageously be placed on different layers. In addition to demultiplexing the incoming data, line cards **1002** can re-multiplex the outgoing data. This may involve optical-to-electronic conversion for the incoming data and electronic-to-optical conversion for the outgoing data.

All of the request processors $RP_{N,0}, RP_{N,1}, \dots RP_{N,K-1}$ receive requests to send packets to the line card LC_N . In one embodiment illustrated in **FIG. 10A**, there is no communication between the layers. There are K input controllers and K output controllers corresponding to a given line card. Thus, each line card sends data to K input controllers and receives data from K output controllers. Each line card has a designated set of input ports corresponding to a given output controller. This design makes the

reassembly of segments as easy as in the earlier case where there is only one layer.

In the embodiment of **FIG. 10B** there are also $J \cdot K$ input controllers, but only J output controllers. Each line card **1012** feeds K input controllers **1020**, one on each layer **1016**. In contrast to **FIG. 10A**, there is only one line card associated with each output controller **1014**. This configuration results in the pooling of all of the output buffers. In embodiment **1010**, in order to give the best answers to the requests it is advantageous for there to be a sharing of information between all of the request processors that govern the flow of data to a single line card. In this way, using inter-layer communications links **1030**, the request processors $RP_{N,0}, RP_{N,1}, \dots RP_{N,k-1}$ share information concerning the status of the buffers in line card LC_N . It may be advantageous to place a concentrator **1040** between each data switch output **1018** and output controller **1014**. Invention #4 describes a high data rate concentrator with the property that given the data rates guaranteed by the request processors, the concentrators successfully deliver all entering data to their output connections. These MLML concentrators are the most suitable choice for this application. The purpose of the concentrators is to allow a data switch at a given layer to continue to deliver an excessive amount of data to the concentrator provided that data from other layers are light during that period. Therefore in the presence of unbalanced loads and bursty traffic, the integrated system of K layers can achieve a higher bandwidth than K unconnected layers. This increased data flow is made possible by the request processor's knowledge of all of the traffic entering each of the concentrators. A disadvantage of such a system is that more buffering and processing is required to reassemble the packet segments, and there are J communication links **1030**.

TWISTED CUBE EMBODIMENT

The basic system consisting of a data switch and a switch management system is depicted in **FIG. 1A**. Variants to increase the bandwidth of the system without increasing the number of input and output ports are illustrated in **FIGs. 9, 10A and 10B**. The purpose of the present section is to show how to increase the number of input ports and output ports while simultaneously increasing the total bandwidth. The technique is based on the concept of two “twisted cubes” in tandem, where each cube is a stack of MLML switch fabrics. A system that contains MLML networks and concentrators as components is described Invention #4. An illustration of a small version of a twisted cube system is illustrated in **FIG. 11A**. System **1100** can be either electronic or optical; it is convenient to describe the electronic system here. The basic building block of such a system is an MLML switch fabric of the type taught Inventions #2 and #3 that has N rows and L columns on each level. On the bottom level there are N rows, with L nodes per row. On each row on the lowest level there are M output ports, where M is not greater than L . Such a switch network has N input ports and $N \cdot M$ output ports. A stack of N switches **1102** is referred to as a cube; the following stack of N switches **1104** is another cube, twisted 90 degrees with respect to the first cube.

The two cubes are shown in a flat layout in **FIG. 11A**, where $N = 4$. A system consisting of $2N$ such switching blocks and $2N$ concentrator blocks has N^2 input ports and N^2 output addresses. The illustrative small network shown in **FIG. 11A** has eight switch fabrics **1102** and **1104**, each with 4 inputs and output addresses. Thus, the entire system **1100** forms a network with 16 inputs and 16 outputs. Packets enter an input port of the

switches **1102** that fix the first two bits of the target output. The packets then enters MLML concentrator **1110** that smoothes out the traffic from 12 output ports of the first stack to match the 4 input ports of one switch in the 3 second stack. All of the packets entering a given concentrator have the same 4 $N/2$ most-significant address bits, two bits in this example. The purpose of 5 the concentrators is to feed a larger number of relatively lightly loaded lines 6 into a smaller number of relatively heavily loaded lines. The concentrator 7 also serves as a buffer that allows bursty traffic to pass from the first stack of 8 switches to the second stack. A third purpose of the concentrator is to even 9 out the traffic into the inputs of the second set of data switches. Another set 10 of concentrators **1112** is also located between the second set of switches 11 **1104** and the final network output ports.

Given that a large switch of the type illustrated **FIG. 11A** is used for 13 the switch modules of a system **100** shown in **FIG. 1A**, there are two 14 methods of implementing request controllers **120**. The first method is to use 15 the twisted cube network architecture of **FIG. 11A** in place of the switches 16 **RS 104** and **AS 108**. In this embodiment there are N^2 request processors that 17 correspond to the N^2 system output ports. The request processors can either 18 precede or follow the second set of concentrators **1112**. **FIG. 11B** illustrates 19 a large system **1150** that uses twisted-cube switch fabrics for request switch 20 module **1154** and answer switch module **1158** in request controller **1152**, 21 and for data switch **1160**. This system demonstrates the scalability of the 22 interconnect control system and switch system taught here. Where N is the 23 number of I/O ports of one switch component, **1102** and **1104**, of a cube, 24 there are N^2 total I/O ports for the twisted-cube system **1100**. 25

Referring to **FIGs. 1A, 11A and 11B** in an illustrative example, a 26 single chip contains four independent 64-port switch embodiments. Each 27

switch embodiment uses 64 input pins and 192 (3•63) output pins, for a total of 256 pins per switch. Thus, the four-switch chip has 1024 (4•256) I/O pins, plus timing, control signal and power connections. A cube is formed from a stack of 16 chips, altogether containing 64 (4•16) independent MLML switches. This stack of 16 chips (one cube) is connected to a similar cube; and so 32 chips are needed per twisted-cube set. All 32 chips are preferably mounted on a single printed circuit board. The resulting module has 64•64, or 4,096, I/O ports. Switch system **1150** uses three of these modules, **1154**, **1158** and **1160**, and has 4,096 available ports. These I/O ports can be multiplexed by line cards to support a smaller number of high-speed transmission lines. Assume each electronic I/O connection, **132** and **134**, operates at a conservative rate of 300 Megabits per second. Therefore, 512 OC-48 optical fiber connections operating at 2.4 Gigabits per second each are multiplexed at a 1:8 ratio to interface with the 4,096 electronic connections of twisted-cube system **1150**. This conservatively designed switch system provides 1.23 Terabits per second of cross-sectional bandwidth. Simulations of the switch modules show that they easily operate at a continuous 80% to 90% rate while handling bursty traffic, a figure that is considerably superior to large, prior art, packet-switching systems. One familiar with the art can easily design and configure larger systems with faster speeds and greater capacities.

A second method of managing a system that has a twisted cube for a switch fabric adds another level of request processors **1182** between the first column of switches **1102** and the first column of concentrators **1110**. This embodiment, control system **1180**, is illustrated in **FIG. 11C**. There is one request processor, MP **1182**, corresponding to each of the concentrators

between the data switches. These middle request processors are denoted by $MP_0, MP_1, \dots MP_{J-1}$. One role of the concentrators is to serve as a buffer. The strategy of the middle processors is to keep the concentrator buffer **1110** from overflowing. In case a number of input controllers send a large number of requests to flow through one of the middle concentrators **1110**, that concentrator could become overloaded and not all of the requests would arrive at the second set of request processors. It is the purpose of the middle processors **1182** to selectively discard a portion of the requests. The middle request processors **1182** can make their decisions without knowledge of the status of the buffers in the output controllers. They only need to consider the total bandwidth from the middle request processors to the middle concentrators **1110**; the bandwidth from the middle concentrators to the second request switch **1104**; the bandwidth in the second switch **1104**; and the bandwidth from the second switch to the request processor **1186**. The middle processor considers the priority of the requests and discards those that would have been discarded by the request processors had they been sent to those processors.

SINGLE-LENGTH ROUTING

FIG. 12A is a diagram of a node of a type used in the MLML interconnects disclosed in the patents incorporated by reference. Node **1220** has two horizontal paths **1224** and **1226**, and two vertical paths **1202** and **1204**, for packets. The node includes two control cells, R and S **1222**, and a 2x2 crossbar switch **1218** that permits either control cell to use either downward path, **1202** or **1204**. As taught in Inventions #2 and #3, a packet arriving at cell R from above on **1202** is always immediately routed to the right on path **1226**; a packet arriving at cell S from above on **1204** is always

immediately routed to the right on path **1224**. A packet arriving at cell R from the left is routed downward on a path that takes it closer to its target, or if that path is not available the packet is always routed to the right on path **1226**; a packet arriving at cell S from the left is routed downward on a path that takes it closer to its target, or if that path is not available it is always routed to the right on path **1224**. If a downward path is available and if cells R and S each have a packet that wants to use that path, then only one cell is allowed to use that downward path. In this example, cell R is the higher priority cell and gets first choice to use the downward path; cell S is thereby blocked and sends its packet to the right on path **1224**. Each cell, R and S, has only one input from the left and one output to the right. Note that when its path to the right is in use, the cell cannot accept a packet from above: a control signal (running parallel to paths **1202** and **1204**, not shown) is sent upward to a cell at a higher level. By this means, a packet from above that would cause a collision is always prevented from entering a cell. Importantly, any packet arriving at a node from the left always has an exit path available to it to the right and often an exit available downward toward its target, desirably eliminating the need for any buffering at a node and supporting wormhole transmission of traffic through the MLML switch fabric.

FIG. 13A is a timing diagram for node **1220** illustrated in **FIG. 12A**. The node is supplied with a clock **1300** and a set-logic signal **1302**. Global clock **1300** is used to step packet bits through internal shift registers (not shown) in cells, one bit per clock period. Each node contains a logic element **1206** that decides which direction arriving packet(s) are sent. Header bits of packets arriving at the node, and control-signal information from lower-level cells, are examined by logic **1206** at set-logic time **1302**.

The logic then decides (1) where to route any packet: downward or to the right, (2) how to set crossbar **1218**, and (3) stores these settings in internal registers for the duration of the packet's transit through the node. At the next set-logic time **1302** this process is repeated.

The data switch with its control system that is the subject of this invention is well suited to handle long packets at the same time as short segments. A plurality of packets of different lengths efficiently wormhole their way through an embodiment of a data switch that supports this feature. An embodiment that supports a plurality of packet lengths and does not necessarily use segmentation and reassembly is now discussed. In this embodiment the data switch has a plurality of sets of internal paths, where each set handles a different length packet. Each node in the data switch has at least one path from each set passing through it.

FIG. 12B illustrates a node **1240** with cells P and Q that desirably supports a plurality of packet lengths, four lengths in this example. Each cell **1242** and **1244** in node **1240** has four horizontal paths, which are transmission paths for packets of four different lengths. Path **1258** is for the longest packet or for a semi-permanent connection, path **1256** is for packets that are long, path **1254** is for packets of medium length, and path **1252** is used for the shortest length. **FIG. 13B** is a timing diagram for node **1240**. There is a separate set-logic timing signal for each of the four paths: set-logic signal **1310** pertains to short-length packets on path **1252**; signal **1312** pertains to medium-length packets on path **1254**; signal **1314** pertains to long packets on path **1256**; and signal **1316** pertains to semi-permanent connections on path **1258**. It is important that a connection for a longer-length packet should be set up in the node before shorter lengths. This gives longer length packets a greater likelihood of using downward paths **1202** and

1204 and therefore exiting the switch earlier, which increases overall efficiency. Accordingly, semi-permanent signal **1316** is issued first. Signal **1314**, which is for long packets, is issued one clock period after semi-permanent signal **1316**. Similarly, signal **1312** for medium length packets is issued one clock period later, and short packet signal **1310** is issued one clock period after that.

Cell P **1242** can have zero, one, two, three, or four packets entering at one time from the left on paths **1252**, **1254**, **1256** and **1258**, respectively. Of all packets arriving from the left, zero or one of them can be sent downward. Also at the same time, it can have zero or one packet entering from above on **1202**, but only if the exit path to the right for that packet is available. As an example, assume cell P has three packets entering from the left: a short, a medium, and a long packet. Assume the medium packet is being sent down (the short and long packets are being sent to the right). Consequently, the medium and semi-permanent paths to the right are unused. Thus, cell P can accept either a medium or semi-permanent packet from above on **1202**, but cannot accept a short or long packet from above. Similarly, cell Q **1244** in the same node can have zero to four packets arriving from the left, and zero or one from above on path **1204**. In another example, cell Q **1244** receives four packets from the left, and the short-length packet on path **1252** is routed downward on path **1202** or **1204**, depending on the setting of crossbar **1218**. Consequently, the short-length exit path to the right is available. Therefore cell Q allows a short packet (only) to be send down to it on path **1204**. This packet is immediately routed to the right on path **1254**. If the cell above did not have a short packet wanting to come down, then no packet is allowed down. Accordingly, the portion of the switch using path **1258** forms long-term input-to-output connections, another portion using paths **1256** carry

- 1 long packets, such as a SONET frame, paths **1254** carry long IP packets and
- 2 Ethernet frames, and paths **1252** carry segments or individual ATM cells.
- 3 Vertical paths **1202** and **1204** carry packets of any length.

4 MULTIPLE-LENGTH PACKET SWITCH

5 **FIG. 14** is a circuit diagram of a portion of a switch supporting the
6 simultaneous transmission of packets of different lengths, and connections
7 showing nodes in two columns and two levels of the MLML interconnect
8 fabric. The nodes are of the type shown in **FIG. 12B**, which support
9 multiple packet lengths; only two lengths are shown to simplify the
10 illustration: short **1434** and long **1436**. Node **1430** contains cells C and D
11 that each has two horizontal paths, **1434** and **1436**, through them. Cell C
12 **1432** has a single input from above **1202** and shares both paths below, **1202**
13 and **1204**, with cell D. Vertical paths **1202** and **1204** can carry either length
14 of transmission. Two packets have arrived at cell L from the left. A long
15 packet, LP1, arrives first and is routed downward on path **1202**. A short
16 packet, SP1, arrives later and also wants to use path **1202**; it is routed to the
17 right. Cell L allows a long packet to come down from the node containing
18 cells C and D, but cannot allow a short packet because short path to the right
19 **1434** is in use. Cell C receives a long packet, LP2, that wants to move down
20 to cell L; cell L permits it to come, and cell C sends LP2 down path **1204** to
21 cell L, which always routes it to the right. Cell D receives a short packet,
22 SP2, that also wants to go down path **1204** to cell L, but D cannot send it
23 down because path **1204** is in use by the long packet, LP2. Furthermore,
24 even if there were no long packet from C to L, cell D cannot send its short
25 packet down because cell L has blocked the sending of a short packet from
26 above.

1 CHIP BOUNDARY

2 In systems such as the ones illustrated in **Figs. 1A, 1D, 1E, and 1F** it
 3 is possible to place a number of the system components on a single chip.
 4 For example in the system illustrated in **Fig. 1E**, the input controllers (ICs)
 5 and output controllers and request processor combined with the output
 6 controllers (RP/OCs) may have logic that is specific to the type of message
 7 that is to be received from the line card. So that the input controllers for line
 8 cards that receive ATM messages might be different than the input
 9 controllers that receive Internet protocol messages or Ethernet frames. The
 10 ICs and RP/OCs and also contain buffers and logic that are common to all of
 11 the system protocols.

12 In one embodiment, all or a plurality of the following components can
 13 be placed on a single chip:

- 14 • the request and data switch (RS/DS);
- 15 • the answer switch (AS);
- 16 • the logic in the ICs that is common to all protocols;
- 17 • a portion of the IC buffers;
- 18 • the logic on the OC/RPs that are common to all protocols;
- 19 • a portion of the OC/RP buffers;

20 A given switch may be on a chip by itself or it may lie on several
 21 chips or it may consist of a large number of optical components. The input
 22 ports to the switch may be physical pins on a chip, they may be at optical –
 23 electrical interfaces, or they may merely be interconnects between modules
 24 on a single chip.

HIGH DATA RATE EMBODIMENT

In many ways, physical implementations of systems described in this patent are pin limited. Consider a system on a chip discussed in the previous section. This will be illustrated by discussing a specific 512 X 512 example. Suppose in this example that low-power differential logic is used and two pins are required per data signal, on and off the chip. Therefore, a total of 2048 pins are required to carry the data on and off the chip. In addition, 512 pins are required to send signals from the chip to the off-chip portion of the input controllers. Suppose, in this specific example, that a differential-logic pin pair can carry 625 megabits per second (Mbps). Then a one-chip system can be used as a 512 X 512 switch with each differential pin-pair channel running at 625 Mbps. In another embodiment the single chip can be used as a 256 X 256 switch with each channel at 1.25 gigabits per second (Gbps). Other choices include 125 X 125 switch at 2.5 Gbps; 64 X 64 at 5 Gbps or 32 X 32 at 10 Gbps. In case a chip with an increased data rate and fewer channels is used, multiple segments of a given message can be fed into the chip at a given time. Or segments from different messages arriving at the same input port can be fed into the chip. In either case, the internal data switch is still a 512 X 512 switch with the different internal I/Os used to keep the various segments in order. Another option includes the master-slave option of patent #2. In yet another option, internal, single line data carrying lines can be replaced by a wider bus. The bus design is an easy generalization and that modification can be made by one skilled in the art. In order to build systems with the higher data rates, systems such as illustrated in **Fig. 10A** and **Fig. 10B** can be employed. For example a 64 X 64 port system with each line carrying 10 Gbps can be built with two

switching system chips; a 128 X 128 port system with each line carrying 10 Gbps can be built with four switching system chips. Similarly 256 X 256 systems at 10 Gbps require 8 chips and 512 X 512 systems at 10 Gbps require 16 chips.

Other technologies with fewer pins per chip can run at speeds up to 2.5 Gbps per pin pair. In cases where the I/O runs faster than the chip logic, the internal switches on the chip can have more rows on the top level than there are pin pairs on the chip.

AUTOMATIC SYSTEM REPAIR

Suppose one of the embodiments described in the previous system is used and N system chips are required to build the system. As illustrated in **Fig. 10A** and **Fig. 10B**, each of the system chips is connected to all of the line cards. In a system with automatic repair, $N+1$ chips are employed. These N chips are labeled C_0, C_1, \dots, C_N . In normal mode chips C_0, C_1, \dots, C_{N-1} are used. A given message is broken up into segments. Each of the segments of a given message is given an identifier label. When the segments are collected, the identifier labels are compared. If one of the segments is missing, or has an incorrect identifier label, then one of the chips is defective and the defective chip can be identified. In the automatic repair system, the data path to each chip C_K can be switched to C_{K+1} . In this way if chip J is found to be defective by an improper identifier label, then that chip can be automatically switched out of the system.

SYSTEM INPUT-OUTPUT

Chips that receive a large number of lower data rate signals and produce a small number of higher data rate signals, as well as chips that

1 receive a small number of high data rate signals and produce a large number
 2 of high data rate signals are commercially available. These chips are not
 3 concentrators but simply data expanding or reducing multiplexing (mux)
 4 chips. 16:1 and 1:16 chips are commercially available to connect a system
 5 using 625 Mbps differential logic to 10 Gbps optical systems. The 16 input
 6 signals require 32 differential logic pins. Associated with each input/output
 7 port, the system requires one 16:1 mux; one 1:16 mux; one commercially
 8 available line card; and one IC – RP/OC chip. In another design, the 32:1
 9 concentrating mux is not used and the 16 signals feed 16 lasers to produce a
 10 10 Gbps WDM signal. Therefore, using today's technology, a 512 X 512
 11 fully controlled smart packet switch system running at a full 10 Gbps would
 12 require 16 custom switch system chips, and 512 I/O chip sets. Such a
 13 system would have a cross sectional bandwidth of 5.12 terabits per second
 14 (Tbps).

15 Another currently available technology allows for the construction of
 16 a 128 X 128 switch chip system running at 2.5 Gbps per port. The 128 input
 17 ports would require 256 input pins and 256 output pins. Four such chips
 18 could be used to form a 10 Gbps packet switching system.

19 The foregoing disclosure and description of the invention is
 20 illustrative and exemplary thereof, and variations may be made within the
 21 scope of the appended claims without departing from the spirit of the
 22 invention.